REFINEMENT OF TEMPORAL CONSTRAINTS IN AN EVENT RECOGNITION SYSTEM USING SMALL DATASETS

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

May 1997

By
Georgios Paliouras
Department of Computer Science

Contents

List of Tables

List of Figures

Abstract

The central aim of this thesis is to develop novel approaches to the representation and the refinement of event recognition models. The event recognition system is viewed as a temporal expert system, which searches for interesting patterns in a stream of temporally indexed data. The format of the input stream is unusual in comparison to standard work on event recognition, such as speech and sound recognition. It consists of time-stamped events, rather than a set of signal properties measured at fixed time intervals. This format has only recently been studied in the area of temporal event recognition.

This thesis proposes a new graphical representation which facilitates explicit modelling of time. The recognition model is a hierarchy of events, each defined as a sequence of subevents. A distinction is made between low-level events, used in the input data stream, and high-level events, defined by the model. Each event definition in the model constrains the duration and temporal association of subevents. This approach naturally handles overlapping events, which have been overlooked in event recognition systems that do not model time explicitly.

Using this graphical representation, a novel method for refining the temporal constraints of a model is presented. The refinement of the model is based on a small training set, consisting of a sequence of low-level events and the high-level events which should be recognised. The small size of the data set does not allow the use of empirical learning methods. Instead, a knowledge refinement approach is adopted, which utilises the original model parameters to guide the refinement process. This approach differs from standard knowledge refinement methods, in that it can handle the temporal aspects of event recognition. Particular emphasis is given to the association of low-level to high-level events – information that is not provided in the data set.

Two modes of refinement are examined in the thesis: full and partial supervision. The former requires the provision of training information for all of the high-level events in the model. This assumption is relaxed under partial supervision, where training information is provided only for the events at the highest level of the hierarchical model. The issue that arises under partial supervision is the correct distribution of the limited training information to all of the events in the model.

The performance of the refinement method is evaluated on a real-world problem: the thematic analysis of the humpback whale song. The song of humpback whales has been extensively studied and analysed in the biological literature and data has been collected, in the form of tape recordings. An event recognition model is derived for the song and the refinement method is applied using a small set of songs. The results of the evaluation are very encouraging, showing that the system is able to improve significantly an initially inaccurate model, even with the use of very limited training data. This result suggests that the method is suitable for structured hierarchical models, such as that of the humpback whale song. Models of this type are used in a wide range of other event recognition tasks, such as fault diagnosis and image sequence analysis.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright Declaration

Acknowledgements

I would like to start by acknowledging the invaluable support of my supervisor, David Brée, who has always been there for me in the past four years. He has been to me a lot more than Ariadne's ball of thread through the maze of our science.

Additionally, I would like to thank friends and colleagues who have contributed in various ways in the making of this thesis. The following names spring immediately to mind:

> Jens Dörpmund, Paul Horton, Hans Jessen, Rosie Lau, Oliver Lemon, Magnus Rattray, Simon Rohrer, Rizos Sakellariou, George Theodoropoulos, Andrew Wright.

I hope the rest will forgive my weak memory. I would also like to express my gratitude to the non-academic staff of the department, who have always responded very promptly to my numerous requests. Amongst these kind people, I would like to thank especially Angela Linton and Janet Boyd.

I am grateful to several of the people involved in research on marine mammal behaviour, for helping me acquiring the humpback whale songs used in this thesis and offering invaluable guidance on the subject. Special thanks to Roger Payne for sharing his vast field experience with me. Other people involved in this part of the work are:

> Chris Clark, Adam Frankel, Jonathan Gordon, Heidi Homuth, Jan Levine, Justin Matthews, Vicky Rowntree, Steve Mitchell, Katherine Payne, Peter Tyack.

Financially, this thesis was supported by Thomson Marconi Sonar and the Engineering and Physical Sciences Research Council (EPSRC). Their help is much appreciated.

Work on a long-term project such as a PhD thesis can get boring and frustrating at times. I have been very lucky to be surrounded by a large number of lively people who have always helped me to recover from such situations. Out of these people, Dimitris Kalles, Ilias Mavroidis and Nikos Sarris, deserve special thanks for being such good friends.

However, most of all I would like to thank my family who will be as happy as I am to see the completion of this work. Especially, I would like to thank my father Αχιλλέα Παλιούρα and my beloved Δήμητρα Λιονάκη, for bearing the cross with me all the way from the bottom of the hill without a single complaint.

Στη μητέρα μου, Παρασκευή Παλιούρα,
αστείρευτη πηγή αγάπης και δύναμης.

To my mother Παρασκευή Παλιούρα.

Part I

Prologue

Chapter 1

Introduction

The task dealt with in the thesis is the refinement of event recognition models using small datasets. The need for event recognition occurs in a variety of real-world problems. This thesis follows the approach of *temporal event recognition*, in which a declarative model of the events to be recognised is provided, together with an explicit inference strategy for recognising events. A novel representation for the event recognition model is proposed, which facilitates efficient event recognition. Furthermore, a novel method for refining event recognition models, represented in this way, is presented. The refining method is the main contribution of the thesis. This chapter presents the event recognition task in more detail, setting the constraints and goals for the thesis.

1.1 Event recognition

Marine biologists have been studying the sounds of whales for a long time. One of the benefits flowing from these studies is the ability to distinguish between different species [12], or populations of the same species [78, 112] and monitor changes in the size of the populations. The sounds emitted by humpback whales have attracted particular attention, because of their interesting song-like structure [79]. By just listening to a recording, experts in the field are able to recognise the humpback whale song and even identify its structural components. In fact this is the method of analysis, which was used in early studies of the song. This approach has a clear limitation, i.e., the inaccuracy of human acoustic perception. It was soon replaced by a more systematic method, in which the spectrogram of a recording was produced and analysed. A spectrogram is a plot of the acoustic signal in the frequency domain. It depicts the frequency components of the uttered sounds, as acquired by a Fourier transformation, against time. Figure 1.1 shows a small piece of such a spectrogram.

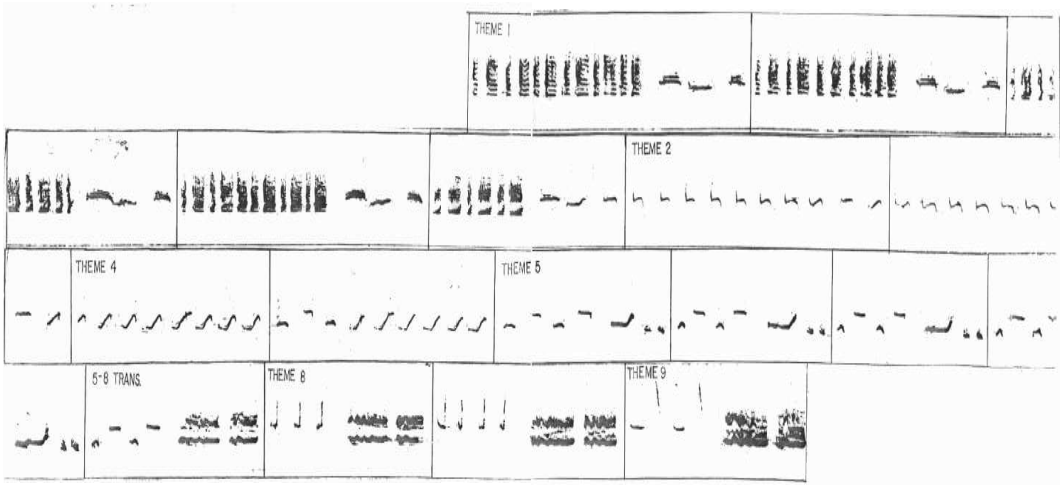


Figure 1.1: An example of a spectrogram from a humpback whale song. Recorded in Hawaii on 7/2/1978.

The visual representation of the sound as a spectrogram allowed scientists to make more accurate measurements of the song components and construct a generic model of the song [79, 76]. This model in turn was used to make other interesting observations, e.g. that the song sung by all individuals in the same ocean basin, at any point in time, is very similar. The model functioned as a set of guidelines, describing how to identify interesting components of the song and compare these in different recordings. In more generic terms, the task of identifying interesting song components in the spectrogram or the recording itself is an *event recognition task*. The events are the song components and recognition is achieved by applying the song model to the recording. An attempt to automate part of this process is presented in chapter 7.

Similar kinds of problem occur in many domains. Examples which deal with acoustic signals are:

Bird-song analysis ([14]). Again, the main task here is to identify the species of the singing bird, which requires the identification of characteristic sounds and structure.

Speech recognition and understanding ([86, 29]). This task involves the recognition of phonemes in continuous speech or isolated word utterances and their combination into syllables, words and phrases.

Classification of musical instruments ([47]). A difficult task for most humans is the identification of different instruments in a musical piece. For experts the task is easier as they are able to associate the properties of the sound with the corresponding instrument.

Underwater sonar classification ([35, 70, 66]). The recognition of interesting underwater events has been of particular interest in military applications.

Fault recognition in airplane turbines ([41]). One way of recognising abnormalities in the operation of turbines is by acoustic monitoring.

However, acoustic signals are not the only domain in which event recognition is required. For example, in the recognition of faults in airplane turbines, several other properties of the operation of the machine can be monitored, e.g. temperature. Although the source of the signal is different, the task remains the same. Other examples are:

Fault recognition in industrial processes ([8]). The task here is similar to that for turbines and there are usually more than one property of the process being monitored.

Recognition of business cycles ([40]). Several aspects of the economy of a country are monitored over time with the goal of recognising interesting phenomena, e.g. business cycles.[1]

Recognition of physical phenomena ([4, 5]). These range from particle collision in a controlled experiment to macroscopic phenomena, such as the explosion of a star.

Event recognition in image sequences ([89]). Examples of these appear in processing of image sequences for autonomous vehicles, robots, etc.

An interesting question is how much the above tasks have in common, i.e., what are the generic features of event recognition. A strict definition of event recognition is not in the scope of this thesis. However, the properties of the task which are of particular interest to the thesis are the following:

- Measurement over time. There is a real-world process, which has observable, time-varying properties, measured as one or more signals. In some tasks there is more than one such processes and more than one property of each process is measured.

[1] A task which has attracted more attention in economics and finance is the *prediction* of events, which can be seen as recognition of events which are to occur in the future. Prediction is based on the recognition of events in the present state of the economy.

The important task feature here is the signal, or signals, which constitute the input to the event recognition system.

- Events. The final aim in the task is the recognition of interesting phenomena, i.e., events, in the data-generation process. For the purposes of the thesis, a distinction is made between *instantaneous* and *durative* events. The former type corresponds to phenomena which happen at a particular point in time, while the latter are associated with a start and end time-point, i.e., they have a duration. An example of the former event type is the failure of a machine in an industrial process or the collision of particles in a physics experiment. Durative events are the utterance of a sentence in a speech recognition problem or a note generated by a musical instrument. Instantaneous events can also have a duration, measured at the appropriate time-scale, but it is not of interest to the task. Similarly, a durative event can be expressed in terms of the instantaneous events of its beginning and end, but its association with the period in which it took place is significant.
- Event recognition model. The recognition of events in a stream of input data is based on a model relating these events to the input signal. In automated event recognition systems, this model can take various forms, e.g. a set of production rules, a neural network or a mathematical model. The model provides a mapping of interesting patterns in the signal, to the corresponding events. Note that in some applications, e.g. industrial fault detection [95], an additional model of the complete process is available, excluding the events of interest. Such a model can help in detecting interesting events, but not in recognising them. This type of detection is not considered in the thesis.
- Inference strategy. This is the method describing how the model is used to recognise events in the data. In some cases, particularly in knowledge-based recognition systems, the strategy is elaborate and independent of the model. In other systems, though, e.g. feed-forward neural networks, it is implicit in the design of the model and it is not considered a separate part of the system.

Figure 1.2 presents a diagrammatic overview of an event recognition system. The system uses an inference strategy to apply the event recognition model to the input signal and recognise interesting events.

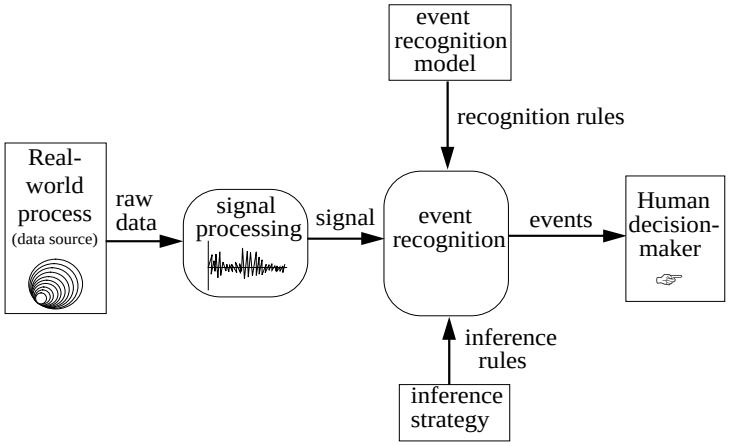


Figure 1.2: Diagrammatic representation of the event recognition process.

1.2 Problem description

This thesis proposes a knowledge-based approach to event recognition, i.e., a declarative representation of the model and a symbolic inference strategy. In this context, a number of assumptions are made about the event recognition task. In particular, it is assumed that:

- The recognition model consists of two separate levels. The first level is a mapping of signal properties to intermediate, *low-level* events. The second is not concerned with the signal at all, but only with the events recognised by the low-level system. It maps the low-level events onto *high-level* events of interest. The motivation for this two-layered architecture is an attempt to capture the expertise involved in the event recognition task in a high-level declarative model. In other words, it is assumed that it is easier for a human expert to describe the mapping between low-level events and high-level ones, than to explain the process of recognising low-level events. Thus, of the two levels only the second is examined in the thesis and henceforth the term event recognition model will be used to refer to this high-level model.
- Events are durative, i.e., the start and end times of events are significant.
- The model is a set of production rules, which contains numeric temporal parameters, constraining the mapping of low- to high-level events. These will be referred to as the model's *temporal parameters* or *constraints*.

Further assumptions and representation details for the event recognition model and the inference strategy are discussed in chapter 3. Figure 1.3 is a diagrammatic description of Fig. 1.2 modified to incorporate the above assumptions.

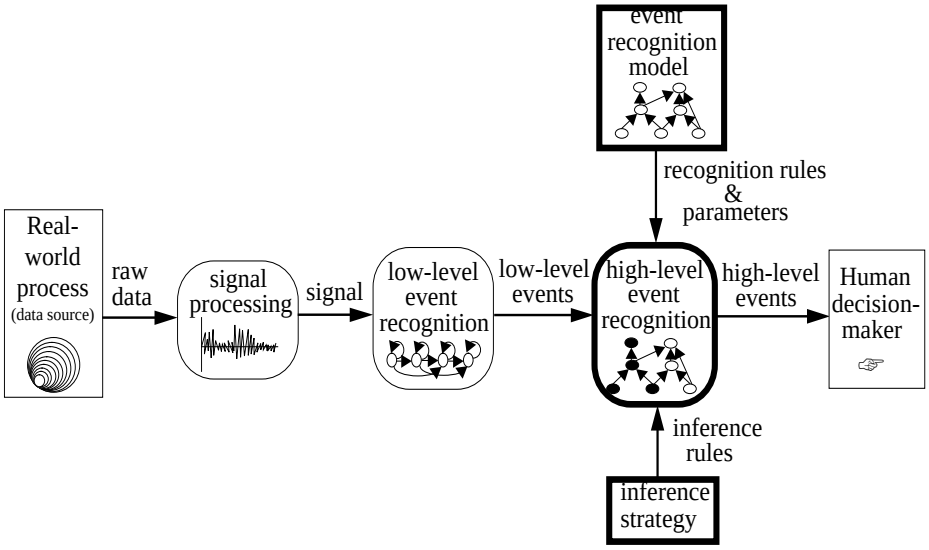


Figure 1.3: Two-level event recognition. The thesis deals with the highlighted parts of the system. The icons in the low-level and high-level event recognition modules correspond to a Hidden Markov Model (discussed in chapter 2) and a Temporal Classification Network, i.e., the representation introduced in this thesis (see chapter 3).

As an example, consider the task of recognising interesting patterns in the song of a whale. Figure 1.4 instantiates the model description of Fig. 1.3 for this task. The low-level recognition module is called a *unit classifier* and outputs a sequence of units, i.e., time-stamped events extracted from the recorded signal. Referring to Fig. 1.1, a unit is a continuous segment of the signal, separated from other units by silence. For instance, at the beginning of the song in Fig. 1.1 there is a sequence of 12 similar units of low fundamental frequency and rich harmonic structure. The module of interest to the thesis is the *theme classifier*, which searches for patterns of units and maps them onto high-level events, called *themes*. In Fig. 1.1, the song is separated into seven themes each of which is a repetition of *phrases*, also identified in the spectrogram. Each phrase has a characteristic unit structure. Thus, the low-level events, i.e., the units, are mapped by the theme classification model onto high-level events, i.e., phrases and themes. Phrases are defined in terms of unit sequences and themes in terms

of phrase sequences. The temporal constraints specify the duration and temporal relations between units/phrases for the recognition of the corresponding phrases/themes. The details of the theme classification task are discussed in chapter 7. Note that the terms *recognition* and *classification* are used interchangeably, because the proposed event recognition system performs recognition by classifying low-level event sequences into the events of interest.

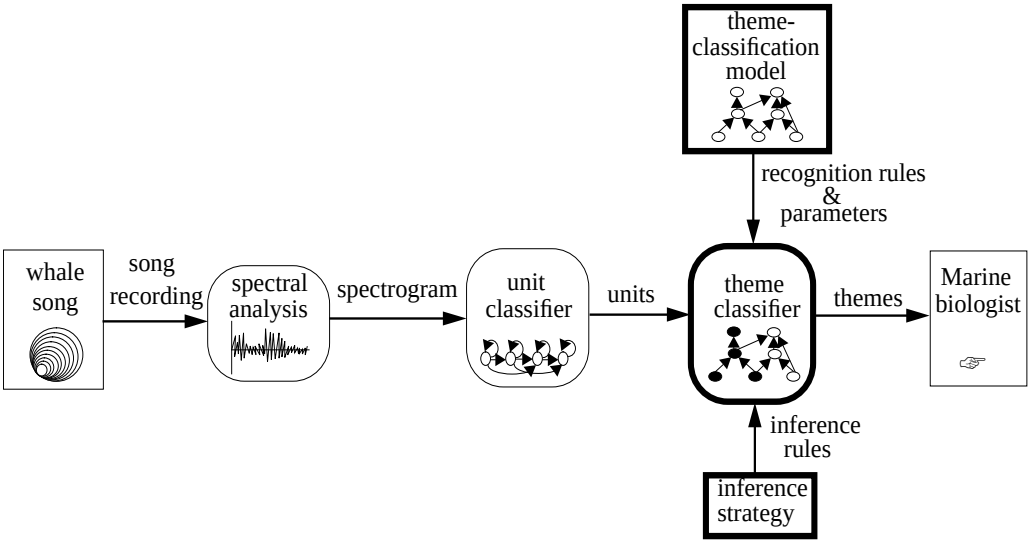


Figure 1.4: Overview of the theme-classification system for humpback whale songs.

Having defined a system of the type shown in Fig. 1.3, it is desirable to be able to modify the event recognition model, in the light of experience. In particular, a supervised learning method is sought, which will modify the model, so as to correctly classify a small set of training data. This process is referred to as *model refinement* and can take two main forms:

- *Model restructuring*, aiming at the modification of the logical structure of the model, e.g. the unit sequence defining a phrase in the whale song.
- *Parameter refinement*, i.e., the modification of the numeric parameters of the model, e.g. the required duration of a unit for the recognition of a phrase.

The focus of the thesis is on the refinement of the temporal parameters. The assumed scenario is that a human expert provides the event recognition model, including approximate estimates of the temporal parameters. The exact determination of the parameter values in a realistic model is a difficult task. For this reason, a refinement method is developed, which automates the

assignment of parameter values. Figure 1.5 augments the system in Fig. 1.3, with the two types of refinement. The modules of the system which are examined in the thesis are highlighted.

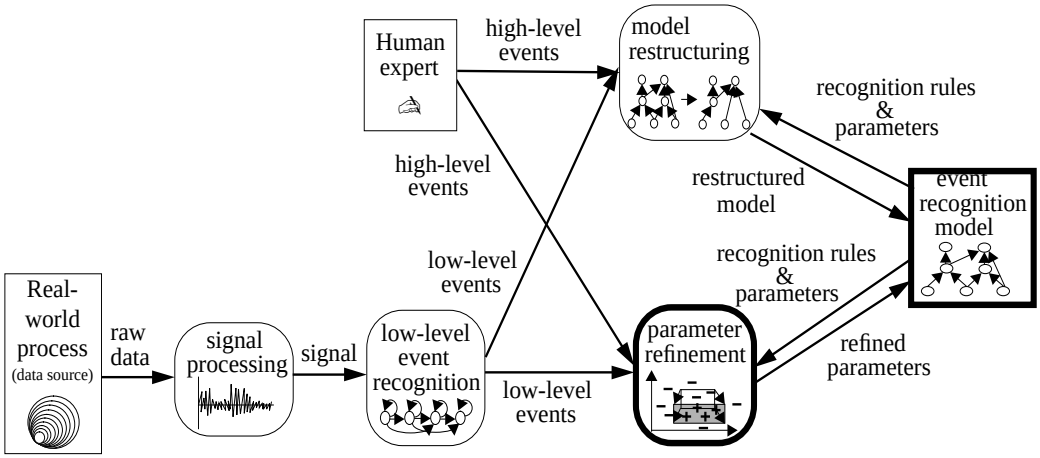


Figure 1.5: Knowledge refinement in the proposed event recognition scheme. The thesis deals with the highlighted parts of the system. The icon in the parameter refinement module represents the modification of parameters in a simple two-dimensional parameter space.

In summary, the thesis aims at providing:

1. a simple rule-based representation suitable for an event recognition model,
2. an inference strategy for recognising events, using a model represented in that way,
3. a method for refining the temporal parameters of the event recognition model, using a small set of data.

The focus and main contribution of the thesis is the refinement method.

1.3 Challenges and constraints

The problem description presented above provides a general outline, ignoring the technical details of the task and focusing on the goals of the thesis. Examining the problem in more detail, a number of challenging issues and problem features come to light, which have restricted the type of solution being sought.

1.3.1 Small training set

All methods which perform learning from empirical data have at least one common characteristic: they aim to construct a robust model which performs well on unseen data, i.e., data that are not included in the training set. This is achieved by the process of generalisation of the training data, which ideally results in a model describing general patterns of interest to the task. The degree to which robust generalisation can be achieved depends on the quality of the data and the generalisation algorithm. The data must be representative of the problem that is solved and sufficient in number to allow general conclusions to be drawn. Thus, the size of the training set plays an important role in the success of the learning process. In many cases, the lack of sufficient training data means that generalisation is not possible.

However, there are several real-world problems where the data are scarce, but useful in modelling the problem. Examples of such event recognition problems are:

- Modelling of natural disasters.
- Turning points in the state of the economy, i.e., the beginning of recession or recovery.
- Failure of the engine of an airplane.

Some of these and other similar problems have been modelled, using expert knowledge about the task, often combined with experimental evidence to refine a theoretical model. The importance of combining *domain knowledge* with empirical modelling has been acknowledged even in early work in machine learning, e.g. [59], where the use of domain knowledge has been suggested, to compensate for the lack of training data. However, domain knowledge comes in various forms and the learning method has to be adapted to the available knowledge. For instance, the available domain knowledge for the humpback whale song may be a set of rules for recognising phrases and themes, but it could also be a stochastic recognition model, such as a Hidden Markov Model (see chapter 2). Also the amount and type of knowledge that is needed to compensate for the shortage of training data depends on the complexity of the learning task, e.g. model restructuring or parameter refinement. *Knowledge refinement* is a field of machine learning, in which domain information is combined with training from data. The motivation for the approach is usually a combination of insufficient data, a complex model to be learned and the existence of useful domain knowledge.

The work presented in this thesis shares the same motivation. The proposed model refinement method performs a type of knowledge refinement, taking into account the peculiarities of

the event recognition task. The basic assumptions are that the type of event which is modelled is rare and that sufficient expertise exists for building an initial model. This model will be inaccurate and will need to be refined by the use of the training data. Once the initial model is available and represented in the appropriate way, it can be used in various ways to guide the refinement method. One such use of the domain knowledge, which is of particular interest to the thesis, is the subdivision of the refinement task in a way that improves the use of the data. This is achieved by a hierarchical decomposition of event definitions, inherent in the representation of the model. Chapter 4 examines in more detail the task of refining an event recognition model using such a representation.

1.3.2 Input data

The input data used by the event recognition system are provided as a stream of events, e.g. units in the case of the whale song, each consisting of a *signature*, i.e., a unique label, and a time stamp denoting its start and end time. For instance, given a set of event signatures $\{A, B, C\}$, the following arbitrary stream of events:

$$B(3,5), A(2,6), B(8,10), C(10,15), \dots ,$$

may be provided as input, i.e., event B from time 3 to time 5, etc., where the measuring scale for time is application-dependent. This format requires a different treatment of time than that assumed by many existing event recognition methods. The usual approach is to use the signal as input to the system and sample from it at regular time intervals, i.e., having an implicit model of time in the system. The reason for not choosing this approach is the two-level event recognition system mentioned above. The advantage is that temporal constraints can be represented as explicit parameters of the model, allowing flexible construction of temporal relations between events. This modelling approach is often referred to as *temporal event recognition*. Chapter 3 investigates this property of the model in more detail.

An immediate result of this input format is that the system is notified about the recognition of an event only when the event ends. Thus, events appear in ascending end-time order. Due to the correspondence between low and high-level events, an event of the latter type is recognised when a corresponding sequence of low-level ones have been recognised, i.e., no sooner than the end time of the last event in that sequence. For instance, if event W is defined as a B

followed by a C, W cannot be recognised before the recognition of $C(10, 15)$. This property of the data affects the choice of inference strategy, as explained in chapter 3.

1.3.3 Overlapping events

Another phenomenon, which is unusual for standard work in event recognition is the use of events which overlap in time. Indeed, overlapping events can be ignored when concentrating on the lower event recognition levels, where a single signal is examined, e.g. low-level speech processing [109, 86]. The separation of the high-level event recognition process from the input signal, or signals, makes this assumption less reasonable. Events may be happening in parallel, recorded on the same or separate signals. This phenomenon appears in many realistic event recognition tasks, e.g. auditory scene analysis [51] and fault recognition in telecommunication networks [25].

Overlapping events introduce an important challenge which affects all stages of the presented work:

- The representation of the event recognition model needs to address the issue of mutual exclusiveness of high-level events. In other words, can overlapping high-level events share sequences of low-level ones? For example, if event Z corresponds to events A and B and event Y to A and C, can both events be recognised in the sequence presented above?
- The model representation also needs to determine the structure of an event sequence and the allowed temporal relations between events. For example, if event Z corresponds to A and B, which of the two possible occurrences of Z, $Z(2, 6)$ or $Z(2, 10)$, should be recognised. $Z(2, 6)$ is recognised using $B(3, 5)$ and $A(2, 6)$, i.e., B is recognised before A. The opposite holds for $Z(2, 10)$, which is recognised using $A(2, 6)$, $B(8, 10)$.
- The assumptions made about overlapping events in the event recognition system affects the complexity of the refinement task, i.e., the search space for temporal parameters which cover the training data.

In order to achieve an efficient event recognition system and a usable refinement method, the type of event that is expected by the system is restricted in chapter 3. However, overlapping events are not excluded, due to their importance in event recognition applications.

1.3.4 Temporal relations

The handling of time is clearly a very important aspect of an event recognition system. As mentioned above, the system developed in this thesis does not have an implicit model of time and allows temporal relations to be defined between events. The natural option for the representation of time in such a system is a temporal logic adapted to the problem at hand. Temporal logics have interesting theoretical properties, but can result in an inefficient system if used in their full generality. In other words, there is a trade-off between expressive power and efficiency of the representation and the balance between these two features is important for a practical system.

The proposed event recognition system makes a number of simplifying assumptions about the type of event that is used and the mapping between low-level and high-level events. Without doubt these assumptions restrict the applicability of the system to a particular class of event recognition problems, which, however, contains a large number of interesting problems. Chapter 3 discusses the assumptions in detail and chapter 7 applies the system to a real-world problem. The immediate benefit from simplifying the handling of time is increased efficiency in event recognition. Additionally, the simplifying assumptions are used in the development of a refinement method for the temporal parameters. Without these assumptions, the parameter space for this relational refinement task would have a very high dimensionality.

1.3.5 Training examples

The training data for the refinement of the event recognition model are not in the format expected by most machine learning methods. In a typical learning problem the training data consist of feature-vector object descriptions and the associated class for the object. This is not the case in the event recognition scenario examined in this thesis. The stream of input events is not divided into examples, i.e., into low-level event sequences associated to high-level events. Thus, the refinement method needs to construct the training examples, using the low and high-level events present in the training set. For the purposes of refinement, the low-level events in the training set are referred to as *input data* and the high-level ones as *feedback data*. For instance, the input data might be the stream of events:

$$A(2,6), B(8,10), B(11,13), C(10,15), \dots ,$$

and the feedback data a parallel stream of the form:

$$Y(2,13), Z(2,15), \ldots ,$$

where Y corresponds to A and B and Z to A, B, C.

This feature of the training data affects the representation of the search space for the refinement task and the refinement method itself. One problem that arises, for instance, in the above example, is that there is more than one sequence in the input data which could correspond to the event $Z(2,15)$, namely:

$$A(2,6), B(11,13), C(10,15),$$
$$A(2,6), B(8,10), C(10,15).$$

Such situations lead to the notion of *alternative positive examples*, discussed in chapter 4.

The construction of negative examples from the training data introduces additional problems, since no negative feedback for misrecognised events is provided. Instead, the set of all examples which could be recognised but do not appear in the feedback data are considered negative. In the above example, the sequence $A(2,6), B(8,10)$ is a negative example of event Y. This weak type of negation increases the complexity of the search space for refinement. The problems appearing in the construction of the set of examples are further discussed in chapter 4. Simplifying assumptions, about the permissible type of event and temporal relation in the system, help in making these problems solvable.

1.3.6 Partial supervision

A fundamental problem in knowledge refinement tasks is that the feedback available in the training data does not cover all parts of the knowledge base. In the context of a production rule system for classification, this means that the teacher provides information only about the terminal items in the inference chain. The correctness of intermediate inferences has to be deduced on the basis of this partial training information. Thus, the question that arises is which part of the inference chain should be modified, if necessary, in order to achieve correct classification of the training data. This question is usually answered by heuristics, which take into account both problem-independent information, e.g. the extent of the required change, and problem-specific information, e.g. the functionality of different rules in the production rule set.

A similar problem arises in Multi-layered Perceptrons (MLPs), where the nodes in the hidden layers do not receive explicit supervision. The most popular solution to this problem is the weighted back-propagation of error, from the output to the hidden nodes of the network. This is made possible by the distributed type of inference that an MLP performs, i.e., the weighted feed-forward combination of support for nodes in the hidden and output layers.

In the event recognition scenario examined in this thesis, the task of refinement under partial supervision is complicated further by the fact that training examples are not explicitly specified in the training data. The additional difficulty is due to the fact that there are usually multiple inference chains, which can lead to the recognition of a high-level event. Chapters 5 and 6 propose methods for refinement under full and partial supervision respectively and examine this problem in more detail.

1.4 Outline of the thesis

Event recognition is a generic task appearing in a variety of real-world problems. In **chapter 2**, several of the influential approaches to event recognition are examined briefly and a few general approaches to knowledge refinement are presented. The properties of the event recognition scenario, as presented in this chapter, are used to argue that most of these approaches are unsuitable for solving the task of interest.

The thesis examines a type of event recognition, in which time is modelled explicitly by temporal relations in the event recognition model. The goal is to provide a method that facilitates the incorporation of domain knowledge in the model, and at the same time allows efficient event recognition. **Chapter 3**, in the second part of the thesis, presents a hierarchical modelling approach, named *Temporal Classification Network* (TCN), which seems particularly suitable to this type of event recognition. The assumptions made by this approach about the type of event and temporal relation that are used are also examined in more detail.

The main focus of the thesis is on the refinement of the temporal parameters in a TCN model. Section 1.3 raised some challenging issues related to this goal, which are addressed by the methods presented in the third part of this thesis. **Chapter 4** presents in more detail the issues and focuses on the representation of the search space, i.e., the construction of candidate modifications to the model. In addition an optimal, but order-dependent, refinement algorithm is presented, which uses a restricted-memory approach.

The next two chapters in this part of the thesis, **chapter 5** and **chapter 6**, extend the memory model, in order to provide an order-independent and noise-tolerant refinement method. These benefits are gained at the expense of heuristic suboptimality and higher computational cost. **Chapter 5** deals with the task of refinement under full supervision, i.e., explicit feedback for all of the events included in the event recognition model. **Chapter 6** relaxes this assumption and presents a method which refines the model with the use of partial supervision.

In the fourth part of the thesis, the methods that have been proposed are applied in practice on an interesting real-world problem. **Chapter 7** describes an event recognition model for the thematic analysis of humpback whale songs. A small set of data are encoded from spectrograms of whale songs recorded in Hawaii, in 1978. This data set is used in **chapter 8**, to evaluate the applicability and performance of the refinement methods. Note that the methods presented in this thesis have not been developed especially for this application and they tackle problems, which do not appear in the humpback whale song data, e.g. overlapping events.

The concluding part of the thesis summarises the experience of developing the methods presented in the thesis, discussing in brief their strengths and weaknesses. Using these conclusions as a starting point, potentially promising paths for further research are sciagraphed.

The appendices to the thesis include the following information:

- Detailed pseudocode descriptions of the algorithms presented in the chapters.
- Extensive proofs for some of the arguments in the thesis.
- Other technical details, e.g. data sources.

Conventions. Each chapter starts with a brief description of the problem that the chapter deals with and the approach which is adopted. This description is separated from the main text and surrounded by a box. Also, the concluding section in each chapter presents a small summary and a critique of the proposed method. Due to the novelty of the methods developed in this thesis, a number of new concepts are introduced, some of which are abbreviated for ease of reading. The first time that a new concept is introduced, it is emphasised in *italics* and, if abbreviated, the abbreviation appears in brackets next to it. Non-standard abbreviations are expanded the first time they are used in each chapter, as a reminder of the concept they represent.

Chapter 2

Alternative approaches

The task of event recognition is encountered in diverse domains. As a result, there is a variety of approaches to the problem, most of which differ significantly from the one proposed in this thesis. The first section of this chapter examines some of these approaches, concentrating on their goals and restrictions. Each approach represents an individual research area with extensive activity. Therefore, no attempt is made to provide an exhaustive account of the work under each topic. The intention is to present an overview of the approaches and show why most of them are inappropriate for the problem of interest. This argument is based on the problem description presented in chapter 1. In a similar manner, the second part of the chapter presents alternative methods for refinement. These methods are not used in the thesis, because they are not suitable for temporal models. However, they might become applicable to the refinement of event recognition models by either extending the methods or reformulating the problem. Neither of these modifications has been considered in the thesis. Thus, the non-suitability of the methods presented in this chapter is conditional on the problem description and the standard description of the methods in the literature.

2.1 Event recognition

The event recognition scenario presented in chapter 1 focuses on one particular view of the problem. Several assumptions have been made about the format of the input data and the properties of the recognition model which are not shared by most approaches to event recognition. The following list summarises these assumptions:

- Symbolic, time-stamped input. The input data is a stream of low-level events and their associated start and end times.
- Overlapping events. Events in the input stream are allowed to overlap on the time axis.

- Explicit modelling of time. The model should facilitate the explicit temporal association of low-level events.
- Variable time scale. The event recognition model may consist of temporal relations which use different time scales. Thus, the inference mechanism should allow the association of events over long and short time periods.
- Efficient event recognition. The structure of the event recognition model should allow the design of an efficient inference algorithm.

The motivation and details of this problem specification are provided in chapter 1 and elaborated in chapter 3. Here, the above assumptions act as criteria for the appropriateness of standard event recognition methods to the problem. Each of the following sections presents briefly an event recognition approach and the restrictions that it imposes on the task.

2.1.1 Hidden Markov Models (HMM)

Hidden Markov models are very popular for modelling stochastic processes and have been successfully used as event recognition systems. One area where they have had considerable success is speech recognition, e.g. [87, 86], which includes several event recognition tasks. For example:

- Classification of isolated word utterances.
- Phoneme or word recognition in continuous speech.
- Phrase identification in continuous speech.

The usual approach in speech recognition is to construct an HMM for each word expected to be uttered and then calculate the degree to which particular word-models match an isolated word, or parts of a continuous speech utterance. HMMs model time implicitly, i.e., by sampling the input signal at regular time intervals.

An HMM can be represented by a State Transition Network (STN), consisting of hidden, i.e., unobservable, states of the process and transitions between them. In speech recognition, the model usually takes the form shown in Fig. 2.1, modelling the progression of speech utterances in time. This type of model is called left-right or Bakis. It has a starting state, i.e., the leftmost one, and does not allow leftward transitions.

An HMM is a doubly stochastic model, modelling two types of uncertainty:

- Each state is associated with a probability distribution over all observable symbols.

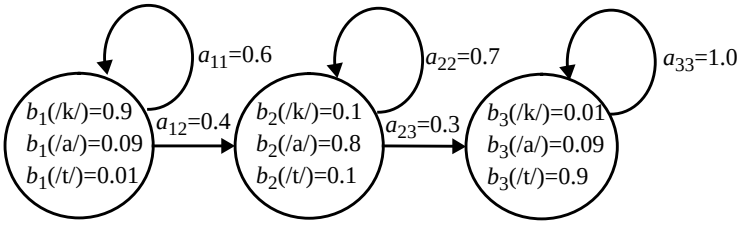


Figure 2.1: A simple Hidden Markov Model (HMM).

In the example of Fig. 2.1 these symbols are the phonemes `/k/,/a/,/t/` and the network models the word `cat`. In other words, each state is not deterministically associated with a unique phoneme, but it is thought to represent a hidden state of the process generating the utterance `cat`. For this reason, HMMs are considered generative models and the probability $b_i(p)$ of phoneme p in state i is called *emission probability*. The emission probability distribution does not need to be discrete. An HMM can be used to model continuous observables and even vectors of continuous variables.

- Each transition between states is also associated with a probability, called the *transition probability* a_{ij}, between states i and j, with the possibility that $i = j$.

In addition to these two probability distributions, a prior distribution over all states needs to be provided for an HMM. In a left-right model the prior probability of the leftmost state is 1 and that of all others 0. As a generative model, a left-right HMM starts from the leftmost state and emits one of the symbols in that state, according to the emission probability distribution. At the next time step it moves to the next state according to the transition probability distribution. This state may be the same as the previous one. Due to the stochastic nature of the model, the generative process can account for different state-transition sequences and different utterance sequences. Each sequence of utterances is associated with a probability, which is calculated as the product of the corresponding emission and transition probabilities.

There are three issues which have attracted considerable attention in HMM research [86]:

- Calculating the match between a model and an observation, e.g. an uttered word. This is needed to classify the observation and involves the calculation of the probability of generating the word by the HMM. An efficient and optimal solution to this problem has been developed, using a dynamic programming technique, called the *forward*

procedure.

- Choosing the state sequence that best explains an observation. This is trivial in the example of Fig. 2.1, but it is a hard problem for more complex models, which allow for pronunciation variations. There are only heuristic approaches to that problem, e.g. the *Viterbi algorithm*.
- Estimating the parameters of the model from data, e.g. from a set of utterances of the word `cat`. This can be considered a learning or model refinement task and there are several approaches to it. The most prominent one is a dynamic programming algorithm, called the *Baum-Welch algorithm*, which performs a maximum-likelihood estimation, learning from positive examples only. An alternative to maximum-likelihood estimation is to use a *maximum mutual information* (MMI) approach, which maximises the discrimination between competing models.

Rabiner [86] describes these approaches in detail, together with real-world applications of HMMs.

HMMs have been criticised for the assumptions that they make about the nature of the problem. The first-order, Markov, assumption is particularly relevant to this thesis. According to this assumption, the transition and emission probabilities depend only on the current state of the model. In other words, the symbol to be emitted at that state and the state in which to move next, are independent of the state sequence and emissions in the past. This assumption leads to difficulties when modelling the temporal properties of events in an event recognition model. For example, duration in an HMM is implicitly modelled by the probability of self-transition in a state. Due to the fact that probabilities are multiplied along a state sequence, the probability of an event decreases exponentially with its duration, e.g. after d self-transitions in the first state of the network in Fig. 2.1, the probability of emitting `/k/` becomes $(a_{11}b_1(/k/))^d$, causing an exponential decrease in the match of the model to the uttered word. This is very unnatural and a number of alternative approaches have been considered, e.g. the imposition of minimum and maximum duration by state duplication, in an acyclic HMM. For long duration ranges this approach can result in very large models, with a large number of parameters.

The focus in HMMs is on the stochastic nature of the event being modelled, i.e., the variability between different event occurrences. This issue is not considered in this thesis. It is

assumed that most of this variability is accounted for by the low-level event recognition system, which could use HMMs. The remaining variability at the higher level of event recognition should be expressible in a symbolic, i.e., rule-based, format. The central issue in the thesis is the incorporation of explicit temporal constraints in the model, which is problematic in HMMs. This is the main reason why HMMs are inappropriate for the examined event recognition task. Moreover, the implicit modelling of time is unnatural for the assumed format of the input data, i.e., time-stamped events, and complicates the representation of overlapping events. Overlapping events are unusual in speech processing, where HMMs are most widely used. The modelling of overlapping events corresponds to the modelling of different voices in a chorus and their temporal dependencies.

2.1.2 Artificial Neural Networks (ANN)

Artificial neural networks have been applied extensively to classification problems, including several types of event recognition. In those areas they have been mostly seen as competitors to HMMs and have often managed to outperform them. The reasons why they have not been considered in this thesis are similar to those mentioned above for HMMs. An additional difficulty that arises in ANNs is the translation of expert knowledge to a distributed inference model as found in an ANN and vice versa. This issue has attracted considerable attention recently, but largely remains an open question. Some attempts to solve this problem are mentioned in section 2.2, which looks at knowledge refinement research. This section provides a very brief overview of ANN approaches to modelling event recognition problems.

Multi-layered Perceptrons (MLP). MLP is the most popular ANN architecture for classification models. An MLP can be graphically represented by a Directed Acyclic Graph (DAG), the nodes of which are organised into layers, usually fully connected. Each node is associated with an activation function, most often a sigmoidal transformation of the weighted sum of its input. The input to the node is provided by the adjacent nodes in the graph. The input layer of nodes corresponds to the representation of the data, e.g. low-level events, and the output nodes provide the classification. Each output node corresponds usually to one class, e.g. one high-level event. Figure 2.2 shows an example of such a network. Due to their widespread use, MLPs were the first ANN models to be applied to event recognition problems, e.g. [92].

The way in which this is usually achieved is by supplying a piece of the signal to the ANN as input and requesting the classification of part of it. The remaining part acts as "contextual information", which is essential for performing event recognition. A similar, static, approach, is to first decompose the signal into events, e.g. words, and supply these as input to the MLP for classification, e.g. [35]. In both cases, the MLP architecture has been criticised as being too rigid for processing time-dependent information, e.g. [52], and has been replaced by more suitable architectures in recent years.

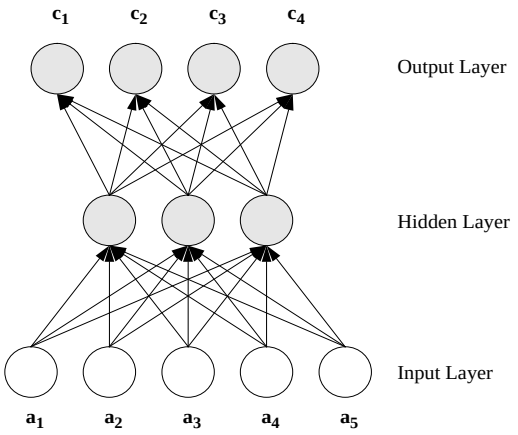


Figure 2.2: A Multi-Layered Perceptron (MLP).

Time-Delay Neural Networks (TDNN). A modified MLP architecture designed to capture temporal speech patterns is the Time-Delay Neural Network [106]. This network consists of two hidden layers, which aggregate information over time frames. Each node in the first hidden layer receives input from a number of adjacent time frames and combines it in a weighted sum, as in the standard MLP. Figure 2.3 shows a case where the input is described by a single feature, e.g. frequency, and four time frames are used as input to the network. A similar process happens at the next hidden layer, which receives the output of the first one over a number of time frames. In Fig. 2.3, the second hidden layer uses the output of the first hidden layer over two time frames. The time scale of the hidden stream differs from that of the input stream. Their correspondence depends on the overlap between consecutive input patterns. If there is no overlap, two time frames in the hidden stream of Fig. 2.3 correspond to eight in the input

stream. The output nodes perform a weighted summation of the output of the nodes in the second hidden layer, using a set of non-trainable weights. This architecture can result in very

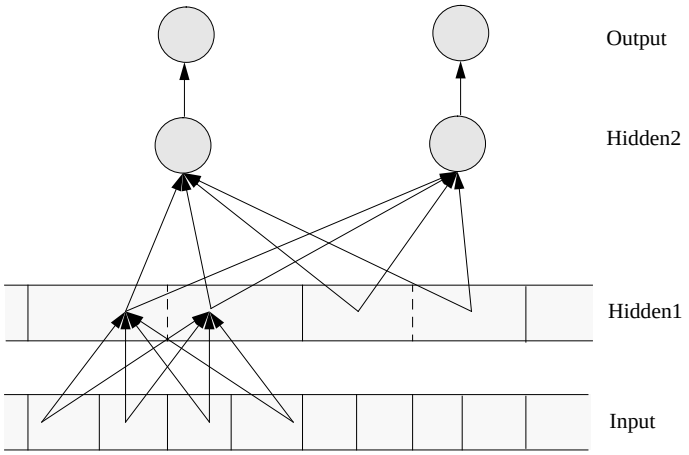


Figure 2.3: A Time-Delay Neural Network (TDNN). The first hidden layer aggregates information over four time frames in the input stream and the second over two frames in the hidden stream. The input stream is one-dimensional and the hidden stream two-dimensional.

large parameter spaces, slowing down the learning rate, i.e., requiring a large training set. In a simple phoneme-classification example, given in [106], $(8 + 3)$ hidden nodes resulted in 504 adjustable weights. However, the use of temporal information in the model improves its performance over the simple MLP. An extended version of the TDNN is incorporated in a large speech-to-speech translation system, called JANUS [105]. Interestingly, JANUS has been used with overlapping speech, by two different speakers, recorded on separate channels. However, this situation can be divided into two separate event recognition problems, which do not need to be combined. Therefore no modelling of overlapping events is necessary. Modelling of overlapping events with a TDNN is problematic.

Recurrent Neural Networks (RNN). Another approach to modelling time with ANNs is the Recurrent Neural Network. An RNN extends the layered architecture of the MLP, with additional links from the output to the hidden nodes. The values generated by the output units are processed as additional input in the next time frame, providing information about the previous state of the process. Figure 2.4 illustrates the structure of an RNN. An alternative to using the values of the output nodes as contextual input is to use the output of the hidden

nodes for the same purpose. One problem with RNNs, which is also a problem of TDNNs, is that they cannot model temporal relations over long periods. As a result, these systems have mostly been used for phoneme recognition, e.g. [109]. The issue of modelling longer temporal relations with RNNs has attracted considerable attention lately, e.g. [6]. Another open question for RNNs is the translation of declarative knowledge to an RNN and vice versa. Efforts in that direction have concentrated in knowledge represented by deterministic [72] and non-deterministic automata [32]. The motivation for this work is the translation of HMMs for speech recognition into a distributed neural network representation. This would allow the use of ANN learning algorithms for this kind of model. Even if this effort were to be fruitful, however, an automaton representation and the corresponding RNN would not be suitable for the problem examined in this thesis, as explained in section 2.1.1.

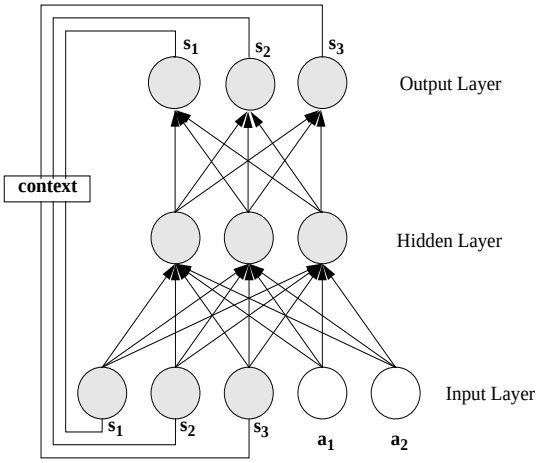


Figure 2.4: A Recurrent Neural Network (RNN).

2.1.3 Knowledge-based signal processing (KBSP)

Hidden Markov models and neural networks are particularly suitable to low-level event recognition tasks, due to their stochastic modelling capabilities. They are less appropriate for high-level event recognition, which involves temporal relations at a larger time scale and the aggregation of information from separate sources. This limitation arises from their implicit modelling of time. An example of a high-level event recognition application is the speech-to-speech translation system JANUS [105], mentioned above. In such a system the usual approach is to

provide knowledge-based models which facilitate the high-level interpretation of the events recognised at lower levels.

One of the earliest and most influential systems which adopted this approach is HEARSAY II [29], which dealt with the problem of speech understanding. In this system a new problem-solving strategy was introduced, called *opportunistic* problem solving, which uses the idea of a *blackboard architecture*. This architecture uses a global data structure, the "blackboard", which allows the communication of a number of specialised tools, called *knowledge sources* (KSs). The KSs perform a range of specific tasks, for low-level, e.g. signal segmentation, and high-level, e.g. syntactic inference, speech processing. The blackboard stores the best current hypotheses about the problem to be solved, e.g. possibly identified sentence, and the KSs continuously check the blackboard, aiming to contribute to the solution. Each time, the most promising, according to a heuristic, KS is selected and applied to the corresponding hypothesis, updating the contents of the blackboard. The problem-solving session ends by another heuristic, which evaluates the quality of the current hypothesis, taking also into account the effort already spent on the problem. Figure 2.5 illustrates this process. The same approach was adopted in another acoustical event recognition problem, namely vessel classification by passive sonar [70].

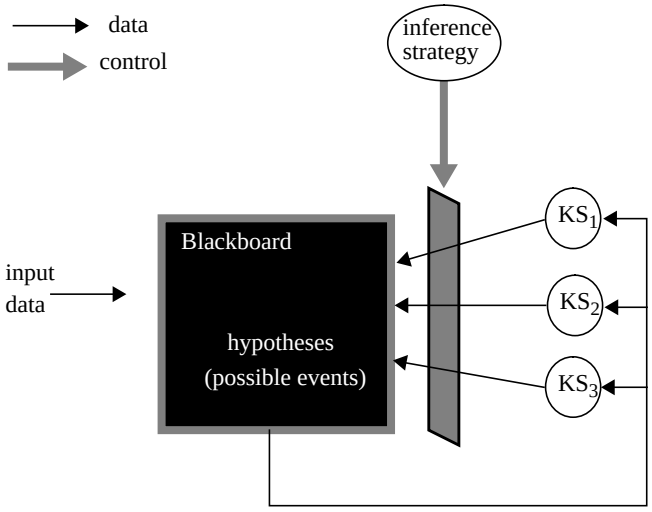


Figure 2.5: A simplified illustration of a blackboard-based system. The inference strategy determines the order in which knowledge sources are applied to the current hypotheses.

Both of the above systems using the opportunistic inference strategy consisted of large knowledge bases and a number of processing elements, i.e., the knowledge sources, dealing with different parts of the inference. This was considered necessary, because of the scale of the problems that were tackled. However, it resulted in very complex systems, suffering in terms of efficiency. The main source of inefficiency is the unstructured problem-solving approach, which needs to consider all KSs at each step of the solution. There have been several attempts to remedy this inefficiency, primarily by structuring the available resources hierarchically and moving through this hierarchy, during the problem-solving session. Examples of this approach appear in [110] and [62].

A recent use of the idea of opportunistic problem-solving appears in the HST (Helicopter Signal Tracker), the components of which are described in [51] and [24]. This is also one of the few examples, where the issue of overlapping events is considered. The context in which this problem is examined is the goal-oriented invocation of appropriate signal-processing algorithms (SPA) for recognising separate events, happening simultaneously and recorded in the same signal. In other words, the aim is the adaptation of SPAs to the current hypothesis in the blackboard, in order to verify and refine it.

The systems described above differ fundamentally from the one proposed in the thesis in two respects: their scope and their applicability. All these systems address a very complex problem, resulting in equally complicated solutions. Such an approach was not possible in this thesis, for which the available resources were limited. This resulted in corresponding restrictions on the scope of the work. One such restriction for example is the exclusion of signal processing and low-level event recognition from the system. Opportunistic problem solving is a very flexible, general-purpose inference mechanism. The specialisation of an opportunistic system to a particular domain, e.g. speech recognition, is done only at the level of the selected knowledge sources. The generality of the approach is also the main source of its inefficiency and the reason why it is not suitable for the problem examined here. The inference mechanism proposed in this thesis, restricts the applicability of the system, in order to achieve efficient event recognition.

The HEARSAY II system, which introduced the idea of opportunistic problem solving, was part of a larger effort to achieve real-time speech understanding. This effort was organised

by ARPA[1], which also specified a list of requirements for the desired system. There was a small number of competing systems developed in this framework, most of which, including HEARSAY II, did not satisfy the ARPA requirements in full. The only successful system was HARPY, which provided a more structured solution to the problem. The HARPY system [53] comprised:

- A syntactic definition of the format of legal sentences.
- A pronunciation dictionary, which described words in terms of phonemes.
- A set of juncture rules, dealing with the detection of word boundaries.
- A template-matching algorithm for the recognition of phonemes.

Each of these components was described in a separate language, but they were all compiled into a state-transition network (STN). This network was the strength and the weakness of the system. It contained all legal phrases in the system, described in terms of transitions between phonemes. The benefit from doing this was an increase in efficiency. The disadvantage was that the resulting network was very large. Lowerre and Reedy [53] mention that the network used for the ARPA exercise contained 15000 nodes, using a restricted syntax and a dictionary of 1011 words. This property of the system renders it inappropriate for unconstrained speech understanding problems.

The HARPY system shares several of its advantages and disadvantages with the system proposed in the thesis. The main disadvantage of both is that all events which can be recognised need to be described explicitly. This can result in large event recognition models, especially for tasks of the type of unconstrained speech understanding. The system proposed in this thesis uses a hierarchical representation of the model, instead of the flat STN used in HARPY, resulting in more compact models with the possibility of alleviating the space-efficiency problem. Further similarities and differences of the two systems are discussed in chapter 3.

2.1.4 Temporal event recognition

The event recognition system proposed in this thesis is best classified under the category of temporal event recognition systems. These systems accept as input a stream of time-stamped low-level events, which the system uses to recognise high-level events of interest. The event recognition model consists of high-level event definitions, which impose temporal constraints

[1] Advanced Research Projects Agency.

on a set of subevents. Temporal logics are used to represent the temporal constraints and constraint propagation algorithms perform the recognition of events.

Work on temporal event recognition can be traced back to the qualitative reasoning system MUSE, presented in [48]. This system allows the definition of qualitative relations between events in the model, using an extended set of Allen's relations for time intervals [2]. MUSE receives input at every time instance and evaluates a set of low-level event prepositions, i.e., it checks whether an event has started or ended at each time instance. In this way, the system can determine the occurrence and duration of an event by its start and end point. This approach runs into problems with events which cannot be recognised before they end. MUSE was applied to a toy blocks-world problem.

More recently, temporal event recognition systems have been applied to real-world problems, such as process monitoring [46] and fault recognition [25]. These systems extend and improve MUSE, allowing quantitative relations and using better constraint propagation algorithms. Chapter 3 examines temporal event recognition systems in more detail and compares them to the system proposed in this thesis.

2.1.5 Other approaches

Besides the above broad research areas, there have been some isolated approaches using different techniques for event recognition. These approaches could be considered variants of KBSP, but what distinguishes them from KBSP is that they use ideas from other research fields. This section provides two examples of such methods.

Rough sets. Rough sets provide an alternative to rule induction from data, based on the idea of redundancy reduction in the examined data set. An interesting introduction to the area is given in [74]. In recent years, rough set construction methods have been applied to signal processing with some success. For example, the performance of a rough-set based speech recognition system has been compared with that of an MLP [19] and the classification of musical instruments by rough set rules has been studied [47]. The way in which these methods are applied is similar to the MLP approaches described above and suffer from the same problems with the modelling of time.

Fuzzy rule sets. There has been at least one effort to use fuzzy rule sets for the recognition of events. The application area is the classification of dolphin sounds and the approach is described in [96] and [3]. The proposed method is a way of post-processing the classification results of a neural classifier, in order to take into account contextual information. The aim of this post-processing stage is to provide more robust classification. Fuzzy rules are used for evidence combination in the system. The system does not attempt high-level event classification and therefore does not make use of temporal relations over a large time scale. It provides, however, an interesting integration of an ANN classifier with expert knowledge to solve the low-level event recognition task.

2.2 Knowledge refinement approaches

The central issue in this thesis is the development of refinement methods for an event recognition model. The model is assumed to adhere to the criteria listed at the beginning of section 2.1. Moreover the refinement method should satisfy the following requirements, as explained in chapter 1:

1. Refinement with a small training set. It is assumed that the number of examples for each high-level event is small, ruling out purely empirical learning methods.
2. Refinement of temporal parameters. The target of refinement is the set of numerical parameters describing the events in the model, e.g. event duration.
3. Construction of the training set. The correspondence between input and feedback data is not provided. The refinement method should establish the mapping between sequences of low-level events in the input stream and high-level events in the feedback stream. This task becomes particularly hard with the use of overlapping events.

Knowledge refinement deals with the issue of combining domain knowledge with empirical data, in order to solve a difficult modelling problem, where the domain theory is inaccurate and/or the training data is sparse. The following sections describe briefly the most influential refinement methods for symbolic models, concentrating on the reasons why they cannot be used here. The main problem is that they are not designed to deal with time-dependent data of the form assumed above. In particular, they assume that the training set is organised as a set of independent examples and therefore do not satisfy the third of the above requirements.

The HMM and ANN approaches to event recognition, presented in section 2.1, facilitate learning of the model parameters from data. In HMMs, the Baum-Welch algorithm and its variants can be used to estimate the stochastic parameters of the model. Given an event, e.g. a word, and the HMM, the parameter estimation establishes:

- the mapping between symbols and the event,
- the associations between symbols and states,
- the most common paths through the STN, corresponding to the event.

However, HMMs are inappropriate for explicit modelling of the temporal parameters. The only way to achieve the desired refinement would be the development of a method which translates the explicit parameters in the implicit model of time used in an HMM. This task becomes particularly difficult when overlapping events are used.

The parameters of ANN models can also be estimated from data. The most commonly used learning method is back-propagation of error. ANNs suffer from the same problems of explicit time modelling as the HMMs. Moreover, the ANN learning methods require the use of a large training set, due to the large number of parameters which need to be estimated. The source of this problem is that learning aims at the construction of the complete event recognition model from data. The initial ANN is a generic description of the parameter space and does not impose constraints on the structure of the event recognition model. The translation of expert knowledge to ANN representations is expected to improve the learning rate of ANNs, i.e., reduce the size of the required data.

2.2.1 Explanation-based learning approaches

One of the earliest attempts to combine domain knowledge with empirical data is an approach called Explanation-Based Learning (EBL). The primary task in EBL is to change the domain theory, in such a way as to make it directly usable in the range of tasks that the system needs to perform. More specifically, in the case of classification, an EBL algorithm uses the theory to generate classifiers, called *concept descriptions*, which are represented in the format of the input data. In order to achieve this goal, the EBL algorithm examines a small number of positive examples – perhaps only one – generating explanations of why, according to the domain theory, they are positive examples of the concept. These explanations are generalised to remove example-specific constraints. The result of this process is a concept description which is a

generalisation of the examples and a specialisation of the domain theory, useful for identifying positive examples of the concept.

The most influential EBL algorithm is Explanation-Based Generalisation (EBG) [65]. Figure 2.6 presents a narrative specification of the algorithm. Central to the EBG formalism is the idea of *operationality*. It expresses the usability of a concept description and it is used to define a stopping criterion for the generalisation of the training example. The initial concept description is assumed to be non-operational, i.e., it is defined in terms of conditions which cannot be directly verified. In other words, the representation of the concept differs from that of the examples. The domain theory in EBG is usually expressed as a set of predicates, each of which defines a subconcept. Some of these subconcepts are found in the definition of the goal concept. Thus, the function of the goal concept description is to draw the attention of the learner to those characteristics of the training example that are known to affect its membership to the concept.

INPUT:

Goal Concept: A non-operational description of the concept to be learned.
[e.g. an arbitrary phrase as a sequence of words.]

Training Example: A **positive** example of the concept.
[e.g. a sequence of time-stamped phonemes.]

Domain Theory: A set of rules and facts, which are used for building the explanation.
[e.g. pronunciation dictionary, syntactic rules, etc.]

Operationality Criterion: A criterion for judging whether a concept description is directly usable.
[e.g. is the sentence expressed in terms of phonemes?]

OUTPUT:
An operational concept description, which is a generalisation of the example and a specialisation of the theory.
[e.g. the sentence as a rule which associates the phonemes in the example.]

Figure 2.6: The EBG formalism, from [65].

The EBG algorithm achieves the operationalisation of the concept description in two stages: explanation and generalisation. In the first stage the theory is used to generate an explanation,

which is an instantiated section of the domain theory and serves as a mapping between the example and the concept. This task is performed by a top-down, deductive theorem prover. In the second stage the generated explanation is generalised. Generalisation involves the replacement of example-specific constants with variables and unification of the variables referring to the same object. The result is a generalised explanation structure, which is a sufficient condition for explaining that the example is an instance of the concept.

The main criticism of EBL is that it does not really perform a learning task. This criticism is based on the concept of *knowledge level learning* [22], which defines learning as the extension of the *deductive closure* of the theory available to the system. The deductive closure of a theory contains the theory itself and whatever can be deduced from it. Since the result of EBL is a specialisation of the theory, i.e., it is included in the deductive closure, EBL does not involve knowledge-level learning. In other words, the system is not able to draw inferences that it could not do before. What an EBL system learns, however, is how to perform certain tasks more efficiently. A similar approach has been adopted in other systems, e.g. the EGGS algorithm [67] and the *chunking* process of Soar [88].

Another important problem in EBL is the assumption that the domain theory and the training example, are "perfect". This assumption, is unrealistic and has been a major topic of research in EBL in the recent years. In a review of EBL research [27] four types of imperfection in the domain theory are identified:

- *Incompleteness:* not covering all positive examples.
- *Incorrectness:* incorrect classification of negative examples.
- *Inconsistency: multiple inconsistent explanations* for the same positive example.
- *Intractability:* no guarantee that an explanation can be derived in polynomial time.

Further to the imperfection of the domain theory, inconsistent or incorrect explanations can result due to noisy training data.

Thus, the EBL algorithm does not perform knowledge refinement. However, there is a number of systems which combine EBL with empirical learning algorithms and they are able to learn from positive and negative examples, aiming to remedy problems in the original domain theory. The following are examples of such systems:

Abductive EBL (A-EBL) [17]. This algorithm deals with the problem of multiple inconsistent explanations for positive examples. The assumed cause of the problem is the overgenerality of the domain theory. Whenever this phenomenon occurs, a selection criterion is applied to choose the best explanation. The EBG algorithm is combined with a greedy set cover method, which uses the negative examples to select efficiently the best explanation. The set cover algorithm is similar to that employed by the AQ-family of inductive learning algorithms, e.g. AQ15 [60], CN2 [16], etc.

FOCL [80, 90]. This system combines an algorithm for inducing simple first-order clauses, FOIL [84], with an inductively guided EBL method. FOIL is a first-order extension of the ID3 algorithm [82], making use of set cover ideas from the AQ family of algorithms. FOIL is being used by FOCL when EBL fails to correctly classify training examples, due to an incorrect or incomplete theory. When this situation occurs, FOIL suggests a number of possible corrections, evaluated by the entropy-based *information gain* criterion.

ML-SMART+EBL [7]. This algorithm incorporates ideas from the inductive learning algorithm ML-SMART, in order to enhance the operationalisation process of its main EBL structure. Similar to EBG, it uses a top-down theorem prover to construct explanations, but improves EBG in many respects:

- It simultaneously constructs more than one explanation.
- Different explanations are evaluated on positive and negative examples.
- It uses a better operationality criterion, proposed in [45].

The above are only a few examples of a large number of systems, which combine EBL with empirical methods to refine the original domain theory. The use of EBL makes this task tractable even with the use of small data sets.

Some of the general concepts underlying these methods are also used in the refinement method proposed in this thesis. An example of such a concept is the use of metrics to evaluate competing modifications to the model. However, the above methods have been applied to static classification problems, where time modelling is not an issue. For instance, the training examples provided to the system are represented in the standard format, i.e., a set of input values coupled with the correct classification. For this reason they are not directly applicable to the refinement of event recognition models.

2.2.2 Knowledge base refinement

The common feature of EBL-based hybrids is the use of the top-down theorem prover to explain the concept membership of positive examples. The knowledge base is treated as a theory and the examples as facts which need to be proven. Alternatives to this approach are grouped here under the category *knowledge base* (KB) *refinement*. The difference between the two approaches is a methodological one rather than one of essence. In KB refinement, the aim is to identify the most likely faults in the knowledge base, by collecting performance statistics on the training set. This approach is designed to be used with large knowledge bases, where theorem proving may be inefficient. The heuristics and cost functions used in the evaluation of individual rules play a very important role in KB refinement.

One of the earliest systems to adopt this strategy was SEEK and its successor, SEEK2 [34].[2] SEEK2 assumes that the knowledge base consists of single-consequence rules, i.e., the conclusion of each rule relates always to a single variable, and that each rule is assigned a *confidence factor* (CF), i.e., a belief on its consequence when it fires. The refinement concentrates mainly on the modification of the CFs of the rules in the knowledge base. This is achieved with the use of heuristics and performance statistics, evaluating individual rules in the knowledge base. Performance statistics are collected by applying the knowledge base on the data and monitoring the behaviour of individual rules. The number of times a rule participates in the misclassification of examples is indicative of its correctness. Heuristics allow the use of explicit biases in the selection of modifications. An important heuristic in that respect is the bias for minimum change, i.e., modifications which incur only small changes to the knowledge base are preferred.

KRUST [18] adopts a similar approach to knowledge refinement. The main difference between KRUST and SEEK2 is that KRUST performs changes to the rule structure, i.e., deletion, replacement and insertion of rules in the knowledge base. Similar to SEEK2 a search for promising modifications is performed, which generates all valid modifications and evaluates them. The evaluation is based on statistical criteria about the performance of each solution on the data. A set of alternative knowledge bases are generated and tested on the data. An important problem with this approach is that the size of the search space increases exponentially with

[2] The main difference between the two versions of the system is that the former is interactive, while the latter automates the refinement process.

the size of the knowledge base. Therefore exhaustive enumeration of the solutions becomes impractical. An improved version of KRUST is incorporated in MUSKRAT [36], which is a tool for knowledge acquisition and refinement. The new version of KRUST uses the idea of "chestnut" cases to reduce the size of the search space for refinement. These cases are selected by the expert as being representative of the problem to be solved.

Another recent example of a KB refinement tool is TGCI1 [23].[3] This system adopts an interesting alternative approach to refinement, integrating the refinement of the initial domain theory with the construction of new features, i.e., change of the problem representation. The original domain theory is expressed as an AND/OR/NOT tree, i.e., a tree in which each node is a conjunction, disjunction or negation of its children. Each example in the training set is evaluated using the original domain theory and the degree of acceptance of the example by the theory is calculated. Partial matches of the example to the theory suggest the construction of new features. The refined theory is a compromise between the two ways of resolving conflicts, i.e., modification of the theory and change of problem representation. The aim is to simplify both the representation of the problem and the domain theory and achieve better classification. This approach faces a similar problem to KRUST, regarding the size of the search space. Therefore an efficient heuristic search for good solutions is essential.

An interesting feature of the KB refinement approaches is the calculation of local statistics for parts of the knowledge base. This process requires a method for assigning credit and blame for misclassification through a possibly complex set of rules. Chapter 6 examines this issue in the context of parameter refinement under partial supervision. The argument for not applying standard KB refinement methods to an event recognition model is the same as for EBL-based methods, i.e., they have not been designed to deal with temporal data.

2.2.3 ANN-based refinement

Artificial neural networks have been widely applied to empirical learning problems with considerable success. The source of their success stems from their ability to estimate highly non-linear models, which are necessary for many real-world problems. The good performance of ANNs has inspired a recent effort to apply them on knowledge refinement tasks. The main obstacle in this effort is the translation of symbolic domain knowledge into a distributed ANN

[3] A more extensive review of KB refinement systems, including some that are not discussed here, is provided in [23].

model and vice versa.

Most of this work has focused on MLP architectures. A representative example is the KBANN system [98], which translates a propositional domain theory into an MLP. Figure 2.7 illustrates this translation. The initial weights in the MLP are selected in a way that reproduces the behaviour of the symbolic theory. The domain theory is an AND/OR/NOT tree similar to TGCI1 described above. The units at different MLP layers correspond to different levels of the theory tree, i.e., the children of a tree node are supporting nodes in the MLP. The initial MLP constructed in that way is not fully connected. Additional connections are added, with small weights, in order to allow the system to learn new dependencies between the variables in the theory. Back-propagation is used for training the network and improving its performance on training data. The translation of the resulting MLP back to a symbolic theory has also been studied, but with less success, e.g. [97].

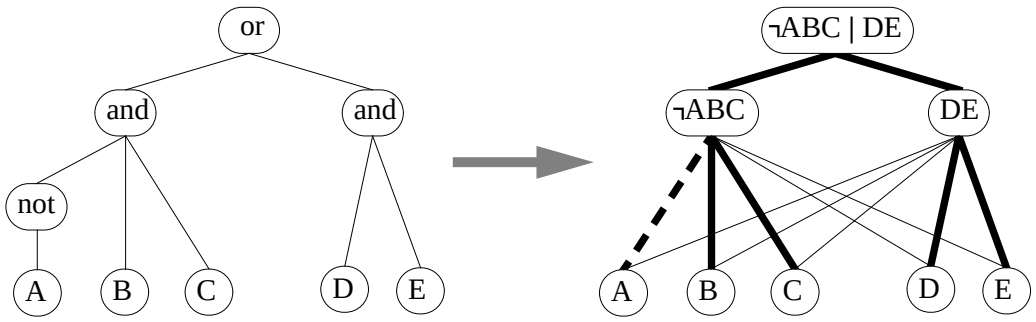


Figure 2.7: Translating a domain theory into an ANN. Strong links are represented by bold lines, weak ones by thiner lines. Dashed lines stand for negative links.

In addition to the work on MLPs, the injection of prior knowledge to RNNs has been studied recently, e.g. [72, 32]. As discussed in section 2.1.2, the use of RNNs for event recognition models suffers from one major problem: the handling of temporal relations over long time periods. This problem is particularly prominent when back-propagation is used for training RNNs. Research on alternative training strategies as well as improved knowledge injection methods for RNNs is currently an active research area, e.g. [6]. The type of knowledge representation, which is mostly considered for injection in RNNs is a finite-state automaton, which is also the representation used for HMMs.

An alternative to the translation of symbolic knowledge to an ANN, is the representation

of symbolic knowledge in a way which allows the use of similar training techniques as in an ANN. Such a representation is the *expert network*, which is studied in [49] and [54]. An expert network is a knowledge base of the form used in SEEK2. The important feature of it is the use of certainty factors in the firing rules, which can be treated in a similar manner as the weights in an MLP. Thus, variants of back-propagation can be used for modifying the CFs in the knowledge base. A further step is to use CFs as an indication for the removal of links and nodes from the network or the addition of new ones. In other words, the modification of the CFs is used as a fault-detection heuristic, in the manner used in the KB refinement methods described above.

The above ANN-based methods are clearly not applicable to the refinement of event recognition models, as examined in this thesis. However the error propagation methods used in ANNs and expert networks provide an interesting alternative to the heuristic credit assignment methods in KB refinement methods. The latter aim at the localisation of blame to individual rules or rule parts. The ANN methods take a global view of the system, due to the distributed nature of the inference mechanism. This difference is examined further in chapter 6.

2.2.4 Refinement of event recognition systems

There are a few exceptional cases, where methods for refining symbolic event recognition models have been developed. Two such cases are briefly discussed here.

Plan refinement. A system for refining plans for speech recognition is described in [21, 20]. In the proposed speech recognition system, plans are used for invoking various knowledge sources of the type described in section 2.1 above. These KSs can be signal processing tools, which extract relevant signal features, or hypothesis formation tools, which help recognising high-level events. The plans are represented in the form of conjunctive production rules and can be modified by examples. Examples are represented by acoustic feature vectors, associated with a class, e.g. a phoneme. The learning method is a simplified version of the AQ rule-induction algorithm [60]. The focus of the learning method is on positive examples, which cause the generalisation of rules, by either introducing rule disjunctions or dropping conditions from conjunctive rules. Rules can also be specialised if they have a large misclassification

rate. Despite the overall temporal nature of the system, the learning task does not involve time-dependent information, allowing the use of standard empirical learning techniques. This is not the case in the problem examined in this thesis.

Genetic algorithms. Montana [66] describes a passive sonar recognition application, using a rule-based system which can be refined with the use of a genetic algorithm. Knowledge refinement concentrates on the certainty factors of rules, which are collected in a real-valued string representing the parameters of the system. Positive and negative[4] recognition examples are used in the optimisation of the parameter string. One problem with this approach is that it ignores the structure of the rule base, treating all of its parameters collectively. This increases the sample complexity of the system, i.e., the amount of training data required.

2.3 Summary

A variety of approaches to event recognition and knowledge refinement have been examined. The event recognition approaches are mainly from the field of speech recognition and understanding. Hidden Markov models (HMMs) are currently the most popular method in this area, allowing the stochastic modelling of various parts of speech. Artificial neural networks (ANNs), in their various forms, are a popular competitor of HMMs. Their popularity results from the use of simple methods for estimating the parameters of the model from data, e.g. back-propagation. Knowledge-based signal processing (KBSP) methods facilitate the integration of various sources of knowledge in large-scale speech and signal understanding systems. The goals and constraints of these approaches differ significantly from the event recognition scenario presented in chapter 1 and for this reason they are not adopted in this thesis. The event recognition system proposed in chapter 3 is an instance of temporal event recognition systems. An important characteristic of this class of systems is the explicit representation of time in event models.

Knowledge refinement is also a very active research area, which has produced a variety of methods for improving knowledge bases with the use of training data. The methods which have been presented here are representative of the activity in the field. EBL-based methods are hybrids of empirical learning methods with the deductive, explanation based learning approach.

[4] Strong negation is used.

EBL focuses the refinement search to the parts of the domain theory used for a particular example and the empirical methods are responsible for the correction of faults in the theory. Knowledge base refinement methods have the same goal, i.e., fault correction, but they adopt a different approach. The rules in the knowledge base are evaluated statistically, using the training set and the most likely faults are identified by a set of heuristics. Finally, a method which has attracted considerable attention in the recent past is the use of ANN-based refinement. This approach requires the translation of symbolic knowledge to an ANN model and vice versa. The main reason why the above methods are inappropriate for the task examined in the thesis is their assumption about the format of the training data, i.e., explicit mapping of the object description to the desired classification.

It is worth stressing that the decision not to use the above methods in this thesis was based on the judgement that they need to be modified significantly in order to be applicable to the problem of interest. This argument does not imply that their use for the representation and refinement of event recognition systems is impossible. Furthermore, one could decide on a different problem representation, which removed the difficult temporal aspects of the problem. Neither of these approaches is adopted here and instead novel representation and refinement methods for event recognition models have been developed.

Part II

Event Recognition System

Chapter 3

A graphical representation for event recognition

The representation of the event recognition model is studied in this chapter. The *temporal classification network* is described, a graphical representation for efficient event recognition which imposes a structured hierarchical modelling of the dependence between events. The representation assumes a symbolic event recognition model, mapping low-level events to the high-level events of interest. Time is modelled by explicit temporal constraints on events, as in other temporal event recognition systems. The types of constraint allowed in the model are discussed in detail. The main contribution of the work presented in this chapter is the structured approach to the design of the event recognition model and the study of event types.

3.1 Event recognition as a temporal classification task

Event recognition entails the classification of input patterns over time into events of interest. Thus, the event recognition task can be seen as a special case of classification in which time acquires particular significance.

The usual approach to classification is to construct a model which maps individual input patterns to classes. The input patterns are presented as vectors of features which are evaluated by the classification model and the class of the pattern is decided.[1] Assuming for example that the task domain is weather forecasting for the following day, some features which might be relevant are the average temperature over the last five days, today's cloud coverage, current wind forces in the region, etc. The classification might be whether it is going to rain tomorrow

[1] In some cases the classes are not mutually exclusive and an input pattern can be mapped to more than one class.

or not. The classification model for this problem may take various forms, e.g. a statistical forecasting model, a neural network or a decision tree. An example, using a rule-based model is the following:

```
IF 5-day-temp < 5°C AND cloud-cov > 70%
OR cloud-cov > 40% AND wind-force < 4
THEN class=rain
```

Note that the classification problem examined in the above example *does* involve time. The temperature is averaged over five days, cloud coverage and wind forces are measured on the current day and the prediction is for the day to follow. However, the representation of the problem removes its temporal aspects and the classifier does not need to model time explicitly. An example of this representational approach in event recognition is the TDNN method for speech recognition, described in chapter 2. This approach is often sufficient for low-level event recognition, where the time-scale involved is small and the number of features which need to be extracted can be limited. However, it does not involve temporal modelling and is not included in the notion of temporal classification as used in this thesis. A temporal classification model for the weather forecasting example would ask questions of the following form:

- When was the last time it rained and for how long?
- Has there been a long period of sunshine in the past month?

Such questions cannot be simply incorporated in a feature-vector representation because they measure weather characteristics over time periods, making time the central aspect of the model.

Event recognition, as examined in this thesis, involves the temporal association of events appearing in the input stream. Sequences of low-level events, which can be appropriately associated in time, are classified to a high-level event of interest. Instead of complicating the representation of the input patterns, by trying to incorporate into it all the relevant temporal aspects of the problem, it seems more practical to include time in the representation of the classification model. Temporal logics, e.g. [2, 104, 93], are the natural choice for such a representation. The strength and weakness of these representations stems from their expressiveness and generality. Temporal logics can be applied to a variety of problems, e.g. planning [56], control [46] and fault recognition [25], but without problem-specific restrictions they can lead to inefficient classification. The representation described in this chapter trades expressiveness for classification efficiency, restricting the model of time to the needs of the event recognition

task.

The event recognition scenario, described in chapter 1, when viewed as a temporal classification task, has the following important aspects:

- The input patterns are simply low-level events, including information about their start and end times.
- Classification is incremental and the temporal order, in which the events appear, is important.
- The classifier consists of a set of rules which determine the temporal relations between events.
- Due to the inherent relation between events, a sequence of input events, rather than a single event, are mapped to the corresponding class, i.e., the high-level event.
- In addition to the classification of an event sequence, the time stamp of the high-level event is determined by the classifier.

As an illustration of this classification process, assume an event recognition task, which uses the signal in Fig. 3.1. The four events marked in the signal, are those recognised by the low-

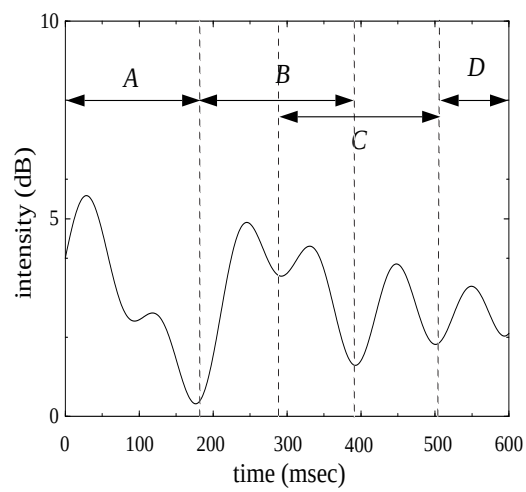


Figure 3.1: An arbitrary signal, with four low-level events.

level event recognition system. The stream of input events for the temporal classifier will be:

$$A(0, 200), B(200, 400), C(300, 500), D(500, 600).$$

A simple classification rule may take the form:[2]

IF $A(i_A^-, i_A^+)$ AND $B(i_B^-, i_B^+)$ AND $i_B^- - i_A^+ = 0$

THEN $Z(i_Z^-, i_Z^+)$, WHERE $i_Z^- = \min(i_A^-, i_B^-)$ AND $i_Z^+ = \max(i_A^+, i_B^+)$.

Using this rule, the high-level event $Z(0, 400)$ will be recognised, as soon as $B(200, 400)$ is received.

The rest of this chapter provides a more detailed description of the proposed temporal representation, the temporal classification network (TCN). First, the nature of time in the representation is discussed. Then the concept of a classification network is presented and proposed as a graphical representation for event models. The temporal aspects of the representation are examined in section 3.4. The use of temporal constraints is restricted to allow efficient event recognition. The event recognition algorithm is described in section 3.5. Finally, the differences of the TCN representation to related event recognition approaches are summarised in section 3.6.

3.2 Time in temporal classification

The underlying nature of time has been the subject of research in many disciplines, e.g. Philosophy, Logic and Artificial Intelligence (AI), resulting in a variety of theoretical and computational models of time. One area, in which most of these ideas are being used, is the field of AI concerned with *temporal reasoning*, where temporal logics are combined with inference mechanisms to build systems that reason about time. Due to their computational orientation, some of the underlying ideas in this work are relevant here. Comprehensive overviews of the work in temporal reasoning are provided in [93] and [103]. The temporal classification representation presented here makes only limited use of the ideas in temporal reasoning. For this reason, an extensive review of temporal reasoning is not given. Instead some of the basic ideas about the handling of time in temporal reasoning systems are presented, augmented with some problem-specific considerations.

The nature of time. Time can be discrete or continuous, linear, branching or cyclic, bounded or unbounded, relative or absolute. The choice of time-definition is usually dependent on the

[2]The notation (i_e^-, i_e^+) denotes the time interval corresponding to the event e. i_e^- is the start and i_e^+ the end of the event.

application. The most common choice for computational systems is for time to be discrete, linear, bounded and absolute. These are the assumptions made in this thesis, too. Discreteness simplifies the complexity of the temporal relations, which use integers, rather than real numbers. Linearity provides the temporal ordering of events, which is necessary for an incremental classification system. The time bounds are not necessary, but they are imposed by the start and end time of the classification session. Similarly, time is absolute because there is an underlying time measurement mechanism which provides time stamps for the low-level events.

Temporal entities. The choice of the basic temporal entity type is the main source of differentiation among temporal reasoning models. The two most common approaches are *time points* and *time intervals*. Point-based systems measure the state of the world at specific points in time, in a snap-shot fashion. This approach resembles that of the speech recognition systems seen in chapter 2, although they are not temporal reasoning systems. An early example of a point-based reasoning system can be found in [56]. Interval-based methods associate their input with time intervals, e.g. the time stamp of an event. Interval-based methods are further subdivided into those which use time-points in the definition of intervals, e.g. [104], and those which use the time interval as the most primitive temporal object. In the latter type the modelling of time is purely qualitative, e.g. [2]. The representation presented here uses time intervals of the former type, i.e., time points underly the definition of the intervals, since the start and end of the interval are discrete time points. This type of entity is common in temporal event recognition systems.

Temporal relations. The repertoire of temporal relations included in a temporal reasoning formalism is dictated to a large extent by the choice of temporal entities. A natural choice when time points are used is the use of linear inequalities between the time points. These can be used even with time intervals when these are defined in terms of their start and end points, e.g. [104, 46]. Using the example of Fig. 3.1, a temporal relation might check whether $i_A^- < i_B^-$, i.e., A starts before B. Three types of relation can be used for time points: $<$,$=$,$>$. Interval-based methods are more complex. An influential qualitative approach for interval-based methods is *Allen's interval algebra* [2], where a comprehensive list of the relations that can be defined between a pair of distinct, finite, convex intervals is presented. Half of these relations are presented in Fig. 3.2. The other half consists of their inverse relations. The

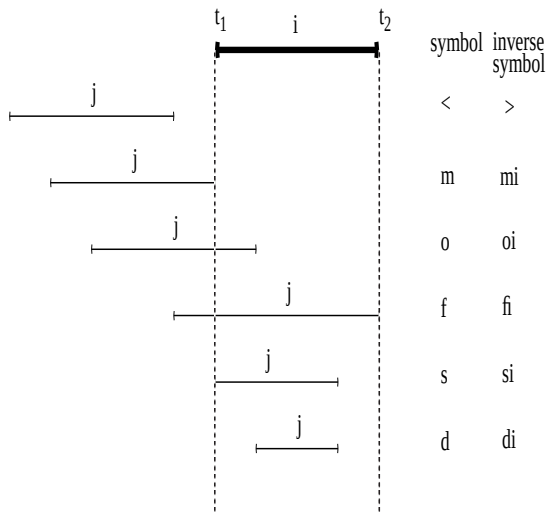


Figure 3.2: Possible relations between a pair of intervals. The symbols assigned to the relations are the ones given in [2] and they stand for the following relations: $<$: before, m: meets, o: overlaps, f: finishes, s: starts, d: during.

approach adopted in this chapter is a quantitative one. Relations are defined between intervals, but can be translated into time-point relations, using the underlying representation of intervals by their start and end points. An example of such a relation would be: $i_B^- - i_A^- > 5$, i.e., B must start at least 5 time points latter than A. This kind of relation has been used in temporal event recognition systems, e.g. [26, 46, 25]. However, the repertoire of temporal relations in the TCN formalism is more restricted than in other temporal event recognition systems, aiming to increase the efficiency of event recognition.

Repeating events. The issue of events which are repetitions of low-order events is of particular relevance to event recognition systems. Using again the example of Fig. 3.1, a repeating event Y might be recognised as a sequence of A's. The representation of the event should allow the specification of sequence length and the temporal aspects of the component events. Repeating or *recurring* events have been paid less attention in temporal reasoning research. Morris *et al.* [68] present a qualitative approach to recurring events, represented by collections of non-overlapping convex intervals. Temporal event recognition systems have not dealt with repeating events. A quantitative approach to the representation of repeating events, in the restricted context of the TCN representation, is provided later in the chapter.

Uncertain time measurement and relations. Uncertainty in the measurement of time and the relations between temporal entities is a problem that every real-world temporal system faces, e.g. [46, 31]. The approach taken here is to use numerical ranges instead of absolute values in the temporal constraints. Any value within the specified range satisfies the constraint.

Mapping event sequences to events. The function of the temporal classification system presented in this chapter is to provide a mapping between sequences of low-level events and individual occurrences of high-level events. This is an important assumption, which seems natural for temporal classification, but cannot be made in most temporal reasoning systems, e.g. planners. This mapping underlies the definition of events which is described in the following section.

3.3 Temporal modelling of events

In the representation proposed in this chapter, events are modelled as sequences of subevents. The subevents of an event are either low-level events, appearing in the input data, or high-level events which are themselves modelled as subevent sequences. This section studies the properties of this hierarchical event modelling approach.

3.3.1 Low-level event recognition

One of the main assumptions in the thesis is that an external system interfaces the raw signal with the temporal classifier. This system performs the recognition of the four events in Fig. 3.1, for example. Although this system will not be examined in the thesis, some of the assumptions that are made about its functionality are important. These are discussed here.

Low-level event recognition may be performed by one of the methods described in chapter 2, e.g. TDNN, HMM, etc., which sample the raw signal at a fixed frequency, measuring several properties of interest, e.g. frequency change and intensity. The elementary time unit may vary and will not be specified here. The information collected in a particular time interval is used to recognise events which have occurred in this interval. In other words, the low-level event recognition system is itself a classifier, mapping patterns of input features to low-level events. It can be described in terms of a set of predicates H^0, each of which corresponds to one low-level

event and has the following form:[3]

$$H^0 = \{h : I \times P \rightarrow \mathbb{B} = \{\mathbb{T}, \mathbb{F}\}\}, \tag{3.1}$$

where $\mathbb{T}$ and $\mathbb{F}$ stand for true and false, $I = \{i | i = (i^-, i^+)\}$ is the set of all possible time intervals, and P the set of input features, i.e., signal properties used in the classification. If E^0 is the set of interesting low-level events, uniquely identified by a label called the event *signature*,[4] there is a one-to-one mapping between the members of H^0 and those of E^0 and the predicate corresponding to an event $e \in E^0$, is indexed by e, i.e., $h_e \in H^0$.

The input features that are used in the classification are not of interest here and can thus be omitted to simplify the definition of the functions in H^0, which now map intervals to events. By applying the predicate $h_e \in H^0$ to an interval $i \in I$, the question which is asked, is whether the event e has occurred, starting at i^- and ending at i^+. If this is the case then $h_e(i)$ is true. The input to the temporal classifier consists of a sequence of occurrences of events in E^0, which take the following form:

$$B^0 = \{e(i) | e \in E^0, i \in I\},$$

where $h_e(i)$ holds. The time interval $i = (i^-, i^+)$ is termed the *time stamp* of the event occurrence and i^-, i^+, the *start* and *end* of the occurrence. The relationship between *event* and *occurrence* is a type-token, or class-instance, one. In the example of Fig. 3.1, A, B are events, $(0, 200), (200, 400)$ are intervals, $A(0, 200), B(200, 400)$ are occurrences with signatures A, B and time stamps $(0, 200), (200, 400)$ respectively.

The input to the temporal classifier does not contain any other information, apart from the event signature and time stamp, but it could be extended to do so. For example an interesting piece of information about an occurrence is the degree of belief in the recognition of the event.

3.3.2 Event recognition function types

In the above discussion and in the rest of the work presented here, the assumption about the event recognition predicates is that they associate occurrences with the *correct* time stamps, i.e., if $h_e(i)$ is true then i provides the start and end time points for the occurrence of e. As a clarification, this does not mean that the time-points are accurate, but they are the best guess

[3]The superfix 0 is used to denote that the examined events are low-level ones.

[4]Note that the use of the term signature here is different than in signal processing.

of the low-level event recognition system about when the event happened. An alternative interpretation, which could be used, is that e has happened some time within i. This alternative is not appropriate for temporal event recognition systems, because the temporal constraints in such systems involve the start and end of events. If these are not known, the validity of the constraints cannot be verified.

A further assumption about the event recognition predicates is that they recognise consistently *all* relevant events. The combination of these two assumptions has an interesting implication to the type of function that can be used for low-level event recognition. In the discussion below, the concept of *subinterval* is needed, which is defined as follows:

Definition 1 *Let $i = (i^-, i^+), j = (j^-, j^+) \in I$. j is a subinterval of i, denoted by $j \subset_I i$, if and only if $i^- \leq j^-$ and $i^+ \geq j^+$ and $i \neq j$.*

Universally subsumed events. There is a class of function for which every subinterval of the interval associated with the event occurrence can be mapped itself to an occurrence of the same event. More formally,

$$h_e(i) \rightarrow \begin{cases} \text{(a) } \texttt{point-interval}(i) \text{ or} \\ \text{(b) } \forall j, j \subset_I i, h_e(j), \end{cases}$$

where `point-interval` is an interval of length 1, i.e., $i^+ = i^- + 1$. Universally subsumed events are the result of ill-defined event recognition predicates of the kind: $\text{intensity}(i) = 10$dB, i.e., all intervals in which the intensity of the signal remains constant at 10dB. Such a function would cause the recognition of the event for *all* subintervals of each interval in which the event recognition predicate holds. Therefore this type of function is not appropriate for defining events and detecting their start and end points.

Self-exclusive events. The other extreme is to disallow self-subsumption of events, i.e.,

$$h_e(i) \rightarrow \nexists j, j \subset_I i, h_e(j).$$

In practice this type of event is very interesting. An example would be to look for maximum intervals of constant intensity. The start and end points of such an event are well-defined and none of its subintervals satisfies the 'maximum interval' constraint. However, there is a theoretical problem with the set of self-exclusive event recognition predicates, namely it is not

closed under the basic logical operators. For example, assuming two self-exclusive predicates h_A and h_B, their disjunction, $h_A \vee h_B$ is not self-exclusive, since an occurrence of A can subsume an occurrence of B and vice versa. A closed set of event types is desirable for the hierarchical representation of events presented in the following section. Note that the use of self-exclusive event recognition functions is not prohibited and in fact it is encouraged, because it reduces the uncertainty in the temporal classification stage.

3.3.3 Recognition time for events

Events in this thesis are strictly durative, i.e., they have a start and end time and their duration is important. Instantaneous events can be represented in this framework by events of duration 1. Strictly durative events are not common in temporal event recognition systems, which usually combine both instantaneous and durative events. In most of these systems, input is received at every time point and the events which have happened at that point are recognised. In other words, low-level events are instantaneous. Durative events are modelled by their start and end points, which are treated as instantaneous events. For instance, in order to recognise the event in which the intensity of the signal is 10dB, an instantaneous event is recognised, when the intensity first takes the value 10dB and another when it changes to a different value.

The two alternative approaches have one fundamental difference: the time of recognition for the events. If instantaneous events underly the recognition of durative ones, an assumption is made that both the start and end of a durative event can be recognised as they occur. The motivation for this approach is on-line and predictive recognition of events, i.e., maintaining at each instance a set of partially recognised events, which have started but not ended. This assumption cannot always be satisfied. For instance, a word in speech recognition cannot be recognised before it ends. The alternative approach of treating events as durative makes the weaker assumption that events can be recognised when they end. Thus, a larger class of events can be represented. This is an important difference of the TCN representation to other temporal event recognition systems.

3.3.4 Hierarchical definition of events

The event recognition model is assumed to be a symbolic one. More precisely, a set of rules, which determine the mapping between low-level and high-level events. In addition, the rules

specify temporal constraints on the relation between the low-level events and determine the time stamp of the high-level event occurrence. The discussion in this section ignores the temporal components of the rules and it concentrates on their structure.

One common approach to rule-based systems is that of production systems [10]. A production system is made of a database, a rule base and an inference mechanism. In the case of the incremental event recognition system, the database must be a dynamic one, i.e., it is updated incrementally by external input. It consists of event occurrences of the form:

$$A(0,200), B(200,400), C(300,500), D(500,600), \ldots$$

The rule base consists of `IF` ... `THEN` rules, which are applied to the database and update it if they fire. An example of such a rule, ignoring the temporal constraints, would be:

`IF` A `AND` B `THEN` Y.

Since A and B occur in the database, Y will be added, with the right time stamp. The left hand side (LHS) of the rule is a logical combination of variables, events in this case, using the basic logical operators. The right hand side (RHS) of the rule may consist of more than one variables, in which case all of them need to be added to the database when the rule fires. Chained inference can take place, in the sense that if there is a second rule:

`IF` C `AND` Y `THEN` Z,

an occurrence of Z is added to the database, when Y occurs. The inference mechanism specifies the order in which rules are evaluated. A common restriction in production systems is that there are no inference loops, i.e. the following rule is not allowed:

`IF` D `AND` Z `THEN` Y.

This restriction is made also here and is referred to as the *acyclicity assumption*.

An important consideration in the design of a production system is the efficiency of the rule-evaluation strategy. This can be particularly critical for an event recognition system, where the database changes incrementally over time. Examples of such production systems are the blackboard-based speech and signal understanding systems mentioned in chapter 2. The approach adopted here for achieving efficient inference is to structure the rule base as a hierarchy of event definitions. In other words, each rule defines an event, which is the variable on the RHS of the rule. If more than one variable appears on the RHS of the rule, the rule can be separated into a number of individual ones containing each of the corresponding variables.

Using the acyclicity assumption, a strict partial order can be imposed on the variables,

i.e., the events, and the rule base can be represented as a Directed Acyclic Graph (DAG) $G = (V, E)$. The vertices, V, of the graph correspond to the events and the edges E to the dependences between them. Figure 3.3 shows an example of such a DAG, using the two rules mentioned above. The relation E defined by the edges of the graph is termed *event support* and

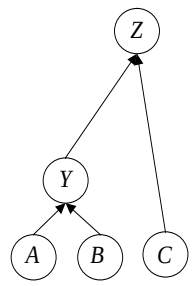


Figure 3.3: Graphical illustration of two event recognition rules.

the partial order represented by the graph *partial support order*. The partial order is denoted by the symbol $<_p$ and is the transitive closure of the support relation E, denoted by $\rightarrow_s$ below:

$$\forall n, m \in V, \quad n <_p m \text{ iff. } n \neq m \text{ and } \begin{cases} \text{(a) } n \rightarrow_s m \text{ or} \\ \text{(b) } \exists p \in V, n <_p p \text{ and } p \rightarrow_s m. \end{cases} \tag{3.2}$$

Using this definition, a partially ordered set V' of events can be generated. Algorithms for producing the ordered set can be found in [102]. In the example of Fig. 3.3 this set is $\{A, B, C, Y, Z\}$. As will be seen below, this ordering of events allows the use of an efficient event recognition strategy.

The partial support order can also be used to define the concept of *order* as a characteristic of events. The low-level events are named 0-order and the events defined on them 1-order. New 2-order events can be defined on 0-order and 1-order ones and so on, building a hierarchy of event definitions. Thus, the order of an event e_k is:

$$\texttt{order}(e_k) = \begin{cases} 0, \text{ if } e_k \text{ is a low-level event or} \\ n+1, n = \max_j(\texttt{order}(e_j)), e_j \rightarrow_s e_k. \end{cases} \tag{3.3}$$

All events e_j, which are used in the definition of a higher-order event e_k, i.e., $e_j \rightarrow_s e_k$, are named *subevents* of e_k. Note that n-order events might include in their definition subevents of

order $< (n-1)$. The sets H^0, E^0 and B^0, corresponding to low-level events, event recognition functions and event occurrences, can be extended to cover the whole set of events in G:

$$\begin{aligned} H &= H^0 \bigcup H^1 \bigcup \ldots \bigcup H^{max}, \\ E &= E^0 \bigcup E^1 \bigcup \ldots \bigcup E^{max}, \\ B &= B^0 \bigcup B^1 \bigcup \ldots \bigcup B^{max}, \end{aligned}$$

where max is the maximum order in the model. In order to be distinguished from the 0-order ones, the term *event definitions* will be used for event recognition functions of n-order, where $n > 0$. The ultimate goal of temporal classification is the recognition of a selected set of special events, which can be of any order (> 0) and are not used in the definition of any other event.

Classification network

The extension of the set of event recognition functions allows another interesting view of the DAG. Since the mapping between events and event recognition functions is one-to-one, the vertices of the graph can be replaced by event recognition functions and the DAG can be seen as a computational, classification network. The input to the network is provided by the 0-order event recognition functions and passed on to the higher-order ones. Classification is propagated through the network in this way and the output nodes, i.e., the events which are not used as a support for other events, provide the final classification. In the cases seen so far, the event recognition functions are simple logical operators. In order to deal with the temporal aspects of event recognition, non-boolean functions should be allowed in a classification network.

This argument can be carried further, broadening the applicability of classification networks to tasks other than event recognition. The generic definition of a classification network is a DAG $N = (F, D)$ the nodes F of which can be defined as a set of functions:

$$F = \{f : J^a \rightarrow O, a \in \mathbb{N}^+\},$$

where the input J and output O sets vary, depending on the type of the function, and a corresponds to the arity of the function. The set of edges D of the graph are directed links between the functions, passing the output of one function as input to others. Input and output to each function may be boolean or numeric and the type of function used in different nodes may vary.

On such a general basis, classification networks can be seen as a generic representation for many other classification systems.[5] Figure 3.4 presents a generic classification network.

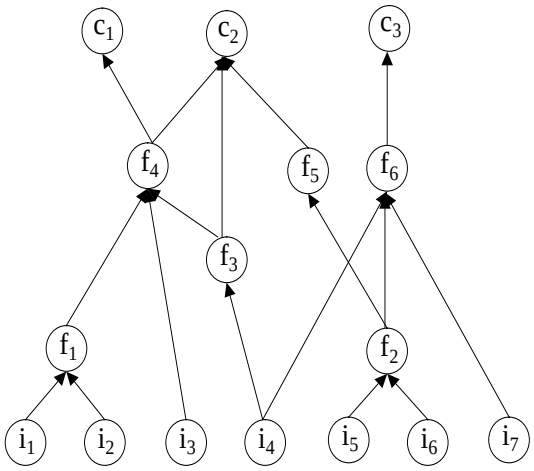


Figure 3.4: A generic classification network. Input nodes are labelled as i_x, classification nodes as c_x and intermediate functions as f_x.

The majority of the classification systems assume a feature-vector representation of the input data and require all 0-order functions to be evaluated each time new input is provided. It is important to note that in an incremental event recognition task this is not the case. The input to the function is accumulated over time and the function can be evaluated when all the necessary input is available. This feature of the problem has an important effect on the design of the database and the inference mechanism, as described in section 3.5.

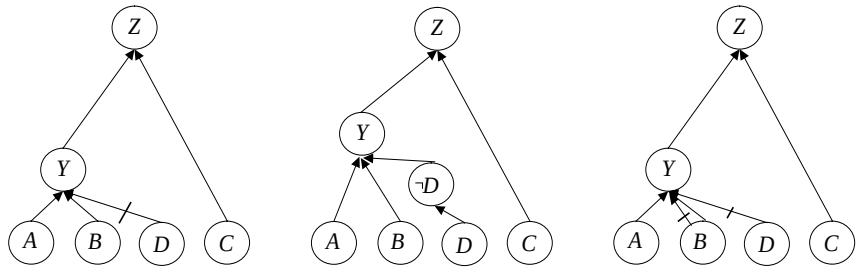
Event and subevent types

There are three types of event definition allowed in the representation: *conjunctive*, *disjunctive* and *repeating*. Negation is represented by defining two separate types of support relation: *supporting* and *rejecting*. Subevents related with a supporting relation to the defined event are called *supporting* subevents. All the definitions seen so far consisted of supporting subevents. *Rejecting* subevents are used to stop the recognition of the defined event. In other words, *supporting* subevents can be seen as positive and *rejecting* ones as negative evidence for the defined event. An example of a rule which contains both types of relation would be:

[5]This can trivially be proven for decision trees and MLP's for example, but it is outside the scope of this discussion.

IF A AND B AND NOT D THEN Y.

Graphically, rejecting links are represented by a crossed edge in the network, as in Fig. 3.5(a). An alternative representation of negation, which does not require the two separate relation types



(a) Rejecting subevent. (b) Negating function. (c) Double link.

Figure 3.5: Representing negation in a classification network.

would be to allow for negating functions as nodes in the classification network, as seen in Fig. 3.5(b). The reason why this approach is not adopted here is because the function would not be a sensible event recognition function any more. It would be true *always*, when the corresponding subevent does not occur. In other words, it would lead to universally subsumed events, which are not practical in event recognition, as seen in section 3.3.2 above.

The use of time stamps allows the distinction between different occurrences of an event, which is ignored in the rules presented so far. This is an important feature of the problem, because it allows multiple input from the same subevent. For example, two separate occurrences of A may be needed in order to recognise Y. The event recognition predicate h_Y, which defines Y, distinguishes between the two occurrences of A, by requiring them to have different time stamps. Moreover, a subevent may be both supporting and rejecting the higher-order event, e.g. one occurrence of B, with some temporal characteristics, may support and another, with different temporal characteristics, may reject the recognition of Y. This can be graphically represented by a double link between the events, as in Fig. 3.5(c).

A repeating event definition consists of a single supporting subevent, which needs to be repeated a certain number of times. For example, event Y^* may be defined as a sequence of occurrences of event Y. The number of required repetitions of Y is allowed to vary within a

range, e.g. $[1..10]$, meaning that 1 to 10 repetitions of Y would cause the recognition of Y^*. By definition, the event recognition function for Y^* makes multiple use of the subevent Y. Such a definition could also be represented as a disjunction of conjunctive events, but this would cause a very large increase in the size of the network.[6]

Consumption of event occurrences

An event of order n can be used in more than one definition of events of order $> n$. This has an implication on the way in which event occurrences are *consumed* in the recognition of higher-order events. For example, if events A, B and C support event Y and events A, B and D event Z, both events Y and Z can be recognised with the use of the same occurrences of A and B. This is not desirable if Y and Z are mutually exclusive. Mutually exclusive classes are the norm in classification systems, but alternatives have been examined, e.g. [59].

In a classification network, complex situations may arise with the use of events which are not mutually exclusive. For instance, events Y and Z may define a higher-order event W. This scenario is shown in Fig. 3.6. In that case W would be recognised using the occurrences of Y and Z, ignoring the fact that these use the same supporting evidence. A possible solution to this would be to flatten the definition of W, requiring the recognition of the full 0-order event sequence, i.e., (A, B, C, A, B, D) to recognise it. This approach is adopted by the HARPY system, briefly presented in chapter 2 (see also section 3.6.2).

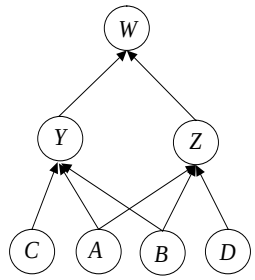


Figure 3.6: Multiple use of subevents.

Flattening the CN is one approach to the consumption of event occurrences which assumes mutually exclusive events. In practical terms this means that low-order event occurrences, e.g.

[6] In this example, 10 extra nodes and 64 extra links would be required.

occurrences of A and B, which are used in the recognition of a higher-order event, e.g. an occurrence of Y, should be removed from the database. The rationale behind this is that if an occurrence of A is associated with an occurrence of Y, it cannot at the same time be used in the recognition of Z. Although this is an intuitive assumption to make, it introduces the problem of choosing among alternative higher-order events, which are supported by the same subevents. An alternative viewpoint is to consider the occurrence of an event as evidence for the occurrence of higher-order events. In that case, the occurrence of A should not be removed from the database when it is used to recognise Y and it could be used in the recognition of Z. If Y and Z are really mutually exclusive, a post-processing mechanism may be needed to decide which of the recognised higher-order events is most likely to account for the underlying sequence of low-order events.

There are arguments for and against each approach. In this thesis, the latter approach is chosen, under the simplifying assumption that the temporal constraints in the model prevent the use of the same event occurrence in the recognition of more than one higher-order events, when this is required, i.e., when the high-order events are mutually exclusive. Thus, in the example presented above, the temporal constraints in the definition of events Y and Z could make it impossible to use the same occurrences of events A and B in the recognition of both Y and Z. Otherwise both Y and Z are recognised. This approach is common in temporal event recognition systems, but it is not necessarily the best option in every application. Speech recognition is an obvious example where problems may arise. This issue requires further study, which is outside the scope of this thesis.

3.4 Temporal relations in an event recognition model

The discussion so far has concentrated on the logical part of the event recognition predicates, ignoring the temporal aspects of the task. This section describes the incorporation of temporal constraints in the event recognition functions.

3.4.1 Temporal association in event definitions

The definition of event e is an event recognition predicate h_e, which combines logically and temporally a set of subevents E_e. The logical part of the function can be seen as a constraint

on the truth values of the subevent predicates and the temporal part is a constraint on the time stamps of the corresponding subevent occurrences. As an example, assume the following event recognition rule:

IF $A(i_A^-, i_A^+)$ AND $B(i_B^-, i_B^+)$ AND $tc(\{(i_A^-, i_A^+), (i_B^-, i_B^+)\})$

THEN $Z(i_Z^-, i_Z^+)$, WHERE $(i_Z^-, i_Z^+) = st(\{(i_A^-, i_A^+), (i_B^-, i_B^+)\})$

In this rule $(i_A^-, i_A^+), (i_B^-, i_B^+)$ are the time stamps of the subevent occurrences, A and B, tc the function specifying the temporal constraint, (i_Z^-, i_Z^+) the time stamp of Z and st the function which calculates it. The following is an example of such a rule, presented in section 3.1:

IF $A(i_A^-, i_A^+)$ AND $B(i_B^-, i_B^+)$ AND $i_B^- - i_A^+ = 0$

THEN $Z(i_Z^-, i_Z^+)$, WHERE $i_Z^- = \min(i_A^-, i_B^-)$ AND $i_Z^+ = \max(i_A^+, i_B^+)$.

In this example the only temporal constraint is $i_B^- - i_A^+ = 0$, i.e., that the end of the occurrence of A should coincide with the start of B. The time stamp for event Z uses the start, i_A^-, of the earlier and the end, i_B^+, of the later occurrence to cover the duration of the subevent sequence, i.e., $(A(i_A^-, i_A^+), B(i_B^-, i_B^+))$. Thus, $(A(0, 200), B(200, 400))$ would cause the recognition of $Z(0, 400)$. The st function does not impose a constraint on the recognition of Z and will therefore be ignored until the end of the section.

The above rule can be expressed in the notation of event recognition predicates as:

$$h_Z((i_Z^-, i_Z^+)) = h_A((i_A^-, i_A^+)) \wedge h_B((i_B^-, i_B^+)) \wedge tc(\{(i_A^-, i_A^+), (i_B^-, i_B^+)\}),$$

where h_Z is the predicate associating Z with the time stamp (i_Z^-, i_Z^+). It would be true for example for $h_Z((0, 400))$. The generic form of the event recognition predicate for conjunctive events is:

$$\begin{aligned} h_e(i) \quad = \quad & h_{e_1}(i_1) \wedge h_{e_2}(i_2) \wedge \ldots \wedge h_{e_n}(i_n) \wedge \\ & \neg(h_{e_{n+1}}(i_{n+1}) \wedge h_{e_{n+2}}(i_{n+2}) \wedge \ldots \wedge h_{e_m}(i_m)) \wedge \\ & tc(\{i_1, i_2, \ldots, i_n, i_{n+1}, i_{n+2}, \ldots, i_m\}), \end{aligned}$$

where $\{e_1, e_2, \ldots, e_n\}$ is the set of supporting subevents for e,

$\{i_1, i_2, \ldots, i_n\}$ the intervals (time stamps) for the supporting subevent occurrences,

$\{e_{n+1}, e_{n+2}, \ldots, e_m\}$ are the rejecting subevents for e,

$\{i_{n+1}, i_{n+2}, \ldots, i_m\}$ the intervals for the rejecting subevent occurrences and

tc the function specifying the temporal constraints on the time intervals.

Note that the subevents in the definition are not necessarily distinct, i.e., it is possible that $e_1 = e_2$. If this is the case, the corresponding occurrence intervals should be different, i.e., $i_1 \neq i_2$. For example, if h_Z requires the recognition of two occurrences of A, $h_A(i_{A_1})$ and $h_A(i_{A_2})$, the two occurrences should be distinct, $i_{A_1} \neq i_{A_2}$. Similarly for disjunctive events:

$$\begin{aligned} h_e(i) &= h_{e_1}(i_1) \vee h_{e_2}(i_2) \vee \ldots \vee h_{e_n}(i_n) \wedge \\ & \quad tc(\{i_1, i_2, \ldots, i_n\}), \end{aligned}$$

where the variables are defined as above. Note that only supporting subevents are allowed in disjunctive definitions. The repeating event definition is an extension of the conjunctive one, excluding rejecting subevents:

$$\begin{aligned} h_e(i) &= h_{e_1}(i_1) \wedge h_{e_2}(i_2) \wedge \ldots \wedge h_{e_n}(i_n) \wedge \\ & \quad w \leq |\{e_1, e_2, \ldots, e_n\}| \leq z \\ & \quad tc(\{i_1, i_2, \ldots, i_n\}), \end{aligned}$$

where $[w..z]$ is the range of required repetitions. A significant difference of this definition from the conjunctive one is that $e_1 = e_2 = \ldots = e_n$, i.e., there is only one event used as a supporting subevent. Therefore, the time intervals must be distinct, i.e., $i_1 \neq i_2 \neq \ldots \neq i_n$. The following discussion will concentrate on conjunctive events. The treatment of the other two event types is similar and only their differences from the conjunctive events are discussed.

3.4.2 Types of temporal constraint

The tc function is itself composite. It is a conjunction of predicates, each of which imposes a constraint on a subset of the involved intervals. Two types of constraint are used in the representation: unary, i.e., involving a single interval and therefore a single subevent, and binary, involving a pair of subevents. In order to avoid confusion about the origin of the constrained intervals, the following discussion assumes that constraints are imposed directly on subevents, rather than their intervals, i.e., occurrence time stamps.

Unary constraints restrict the duration of the subevents. The notation used henceforth for the duration constraint is:

$$\texttt{duration}(e_k, e_j, d),$$

where e_k is the event being defined, e_j the subevent and d the duration constraint. In its simplest form, d is a positive integer corresponding to the length of the time interval associated

with the occurrence of e_j. For example, $\texttt{duration}(Z, A, 200)$ requires that the duration of A in the definition of Z is 200. Thus, the occurrence $A(0, 200)$ would satisfy the constraint. When the duration constraint is a single integer value, it is termed *plastic*. The proposed model extends the duration constraint to allow for uncertainty in the required duration for the event occurrence. Instead of d being a constant value, a numeric range $d = [d^- ..d^+]$ is used and all integers in the range satisfy the constraint. Thus, $\texttt{duration}(Z, A, [180..220])$ is satisfied by $A(0, 195), A(50, 230)$ and $A(50, 270)$. This type of duration constraint is called *elastic*. Each of the two limits of the constraint range is allowed to be undefined, denoted by the symbol '?' for the upper limit and by the minimum duration, 1, for the lower limit.

Binary constraints specify the relative occurrence of subevents. There are various choices for the relational temporal constraint and the most appropriate one will depend on the application. In some applications more than one type of relation may be needed. The relational constraint proposed here makes several restrictive assumptions about the order of subevents in an event definition, in order to achieve efficient classification. An important requirement is that subevents are allowed to overlap and even be completely subsumed by each other. Clearly binary constraints are not applicable to disjunctive events, since each of their supporting subevents is treated individually, i.e., a single subevent occurrence can cause the recognition of the higher-order event.

The first assumption which has already been made above is that the constraints are binary, i.e., only two subevents participate in each relation. Each constraint restricts the occurrence of one of the subevents, i.e., its time stamp, relative to the occurrence of a second one. The following general notation will be used for the binary constraint:

$$\texttt{precedes}(e_k, e_h, e_j, r),$$

where e_k is the event being defined, e_h and e_j, $h \neq j$, two subevents in its definition and r is a numeric constraint on the time stamps of e_h and e_j. Note that the two subevents may have the same signature, i.e., $e_h = e_j$, but their intervals, i.e., time stamps, must be different. As the name of the relation implies, the assumption is made that e_h has to be recognised before or at the same time as e_j, i.e., the end time point of the occurrence of e_j is greater than or equal to that of e_h. This is also the first constraint imposed on the two subevents. If it is not satisfied, the recognition of e_k fails. This is a weak qualitative constraint on the subevent intervals, corresponding to a disjunction of the six interval relations in Allen's algebra (see section 3.2),

excluding their inverse relations and adding the equality[7] relation. If the qualitative binary *precedes* relation is defined as the equivalent of the `precedes` relation for time intervals, then for intervals $i_1, i_2 \in I$:

$$(i_1 \textit{ precedes } i_2) = (i_1 \{<, \mathrm{m}, \mathrm{o}, \mathrm{f}, \mathrm{s}, \mathrm{d}, =\} \, i_2)$$

where ($<$, m, o, f, s, d) are as presented in Fig. 3.2.

Finally, the st function, used in the calculation of the time stamp for the defined event, is simply chosen to cover the whole sequence of subevents in the definition. For example, given event occurrences $A(0, 200)$ and $B(200, 400)$, event $Z(0, 400)$ will be recognised. For this reason, two of the subevents are of particular importance, because they determine the duration of the recognised event. These are the *initial* subevent, i.e., the one with the earliest start point, and the *terminal* subevent, i.e., the one with the latest end point. Using the properties of the `precedes` relation, the choice of terminal subevents can be limited to those which do not precede any other subevents. The same is not possible for the initial subevent, since the subevent which is recognised first is not necessarily the initial one. A subevent which is not preceded by any other may be completely contained in one of the subevents that it precedes. Due to their special status, potentially terminal subevents are used to trigger the inference mechanism. No processing takes place when the incoming event occurrences are non-terminal.

3.4.3 Temporal distances

The addition of the quantitative constraint r restricts the relation between the two intervals further. For example, it may require that event A ends 10 time units before event B starts, in order for Z to be recognised. Using again the general form of the `precedes` relation:

$$\texttt{precedes}(e_k, e_h, e_j, r),$$

the quantitative constraint uses the second subevent, e_j, as the *reference point* on the time axis to constrain the occurrence of the earlier event, e_h. Since each event occurrence is associated with a time interval, rather than a single point in time, there are two time points which can be used for *reference*, i.e., the start or the end of e_j. Similarly, two *offset* points, i.e., the start and end of e_h, can be constrained by the relation. The definition of r chosen here specifies the

[7] Only for different subevent signatures.

length of the interval between the end of the earlier, e_h, and the start of the later, e_j, subevent. This relation is called the *temporal distance* of the two subevents. Figure 3.7 presents example distances between the two intervals corresponding to e_h, e_j.

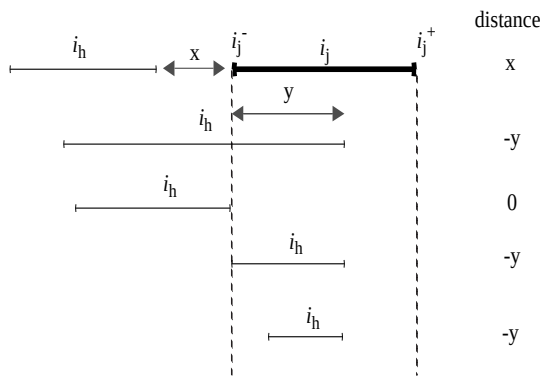


Figure 3.7: Examples of temporal distances.

The qualitative relation between the intervals differs, depending on the choice of distance value and the duration of the intervals. Negative distances represent overlapping intervals. In order to guarantee consistency with the qualitative `precedes` relation, the value of r must be greater than the duration of the preceded subevent, i.e., e_h is not allowed to overlap e_j, by more than e_j's duration.

As with the duration constraint, in the simplest case the distance constraint is plastic, i.e., r takes an integer value specifying exactly the distance between the two subevents. For example, the distance between events A and B in the definition of event Z might be required to be exactly 10 time units. In that case occurrences $A(0, 200)$ and $B(200, 400)$ would not satisfy the constraint and Z would not be recognised.

Plastic distances are too rigid and are expected to be inapplicable to practical event recognition problems. Usually the exact relation between subevents will not be known and/or there will be uncertainty in the measurement of the start and end points by the low-level event recognition system. For that reason, elastic temporal distances are introduced here; they require the temporal distance between two subevents to fall within an integer range. Thus, if the temporal distance between A and B is required to be in the range $[0..20]$ time units, $A(0, 200)$ and $B(200, 400)$ will cause the recognition of $E(0, 400)$. Similar to the `duration` constraint, the range limits for elastic distances can be undefined, taking the value '$?$'.

The number of temporal relations permitted in an event definition plays an important role in the efficiency of the event recognition scheme, as each constraint needs to be checked in the evaluation of an event recognition predicate. For this reason, the following restriction is imposed on the use of the `precedes` relation:

$$\forall e_h, e_j \in E_{e_k}, \texttt{precedes}(e_k, e_h, e_j, r_1) \rightarrow \begin{cases} \text{(a) } \nexists e_l \in E_{e_k}, \texttt{precedes}(e_k, e_h, e_l, r_2) \text{ and} \\ \text{(b) } \nexists e_m \in E_{e_k}, \texttt{precedes}(e_k, e_m, e_j, r_2), \end{cases}$$

where e_h, e_j, e_l, e_m are subevents in the definition of e_k. In other words, a subevent can only be `preceded` and can only `precede` a single other subevent.[8] As an example of the imposed restriction, assume that event Y is defined as a sequence of three subevents A, B, C. If the relation $\texttt{precedes}(Y, A, B, r_{AB})$ is used, the allowable additional relations are:

$$\texttt{precedes}(Y, B, C, r_{BC}) \text{ or } \texttt{precedes}(Y, C, A, r_{CA}),$$

defining respectively the sequences (A, B, C) and (C, A, B). The other possible constraints:

$$\texttt{precedes}(Y, A, C, r_{AC}) \text{ and } \texttt{precedes}(Y, C, B, r_{CB})$$

are not allowed, because A already `precedes` B. The effect of this restriction on the use of temporal distance relations is that it imposes an ordering on the subevents in the definition. If the maximum number of relations, $n - 1$ for n subevents, is defined, the sequence in which the temporally related subevents have to be recognised is fully specified. This restriction will henceforth be referred to as the *precedence sequence assumption* and the resulting sequence of subevents as the *precedence sequence* of the event definition.

The network in Fig. 3.8 corresponds to an event definition with a *single* precedence sequence of supporting subevents. This is not required by the assumptions made so far. Thus, an event definition could contain two or more, temporally independent, sequences of subevents. In that case, it may be sensible to break the event definition into separate events, each consisting of a single precedence sequence. The reason is that the sequences are not temporally related and can be combined in an arbitrary way, leading to the recognition of a large number of occurrences of the defined event. Thus, the situation of more than one subevent sequences in an event definition is considered a rare degenerate case here, but it is not prohibited. The

[8]Note that these restrictions apply to the use of the `precedes` predicate and they are independent of the transitive property of the qualitative *precedes* relation between the subevent intervals, which still holds.

practical implication is that there can be more than one *potentially terminal* subevent in an event definition. Each of them will trigger the evaluation of the event recognition predicate.

The use of rejecting subevents in an event definition presents a further problem for the relational `precedes` constraint. An event definition which contains rejecting subevents imposes the requirement that these subevents do not occur, within the specified temporal constraints. The use of duration constraints on the rejecting subevents is not a problem. However, it is also necessary to determine their occurrence relative to other subevents in the definition. For this reason, they cannot be excluded from the precedence sequence. They can also not be `preceded`, because a legal sequence of subevents should not contain any occurrence of rejecting subevents. In other words, their event recognition function is required to be false and therefore it is not associated with an occurrence interval when the high-order event is recognised. So, rejecting subevents cannot be used as reference to determine the relative occurrence of other subevents. Under the precedence sequence assumption, only one rejecting subevent can be used in an event definition and this has to be related to the first supporting subevent in the precedence sequence. This seems unreasonable and a modification to the precedence sequence assumption has been decided, which provides special treatment of the rejecting subevents. According to the modified assumption, a supporting subevent can be preceded by one other supporting subevent and one or more rejecting ones. Rejecting subevents cannot be preceded and have to precede a single supporting one. The modified assumption implies that each event definition contains a sequence of supporting subevents, onto which rejecting ones may be attached. Figure 3.8 illustrates this by a *precedence network* in which crossed arrows represent precedence by rejecting subevents.

In repeating event definitions the exact number of subevents is not known and they can therefore not be treated individually. To do so would be unintuitive anyway, since the idea underlying repeating events is that they correspond to a sequence of repeating subevent occurrences with similar properties. For this reason, repeating event definitions contain a single `duration` and `precedes` constraint, determining the duration of the subevent occurrences and the temporal distance between them. For example, if event Z^* is defined as $[1..10]$ repetitions of Z, the following constraints need to be defined:

$$\texttt{duration}(Z^*, Z, d_Z), \texttt{precedes}(Z^*, Z, Z, r_{ZZ}).$$

The constraints are still elastic, i.e., d_Z and r_{ZZ} are integer ranges.

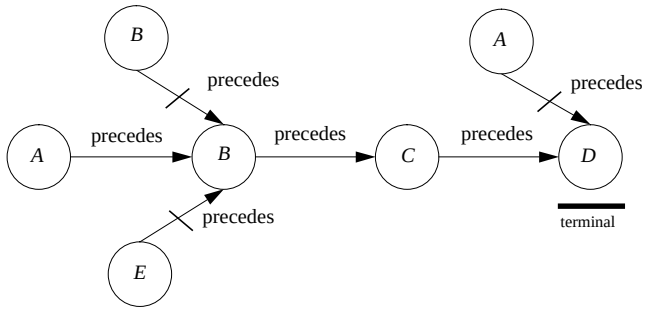


Figure 3.8: A precedence network under the modified precedence sequence assumption. Events C and D act as supporting subevents, event E as rejecting and events A, B as both supporting and rejecting subevents in the definition.

An important effect of the use of elastic temporal distances is the possibility of recognising multiple occurrences of the same event which share some subevent occurrences. For example, assume the definition of event Z as a sequence of A, B, C and the following `precedes` and `duration` constraints:

$$\texttt{precedes}(Z, A, B, [-10..10]), \texttt{precedes}(Z, B, C, [-10..10]),$$
$$\texttt{duration}(Z, A, [8..12]), \texttt{duration}(Z, B, [8..12]), \texttt{duration}(Z, C, [8..12]).$$

Suppose also that the following sequence of event occurrences is recognised:

$$A(0, 8), A(4, 12), B(14, 24), B(18, 26), C(25, 35)$$

This sequence can lead to the recognition of two occurrences of Z, each of which can be recognised in two different ways:

$$\begin{aligned} A(0,8), B(14,24), C(25,35) &\rightarrow Z(0,35), \\ A(0,8), B(18,26), C(25,35) &\rightarrow Z(0,35), \\ A(4,12), B(14,24), C(25,35) &\rightarrow Z(4,35), \\ A(4,12), B(18,26), C(25,35) &\rightarrow Z(4,35). \end{aligned}$$

As mentioned in section 3.3.4, the recognition of both occurrences of Z is allowed. The fact that each occurrence can be recognised in two different ways has an important consequence on the design of the refinement algorithm (see chapter 4).

3.5 Temporal classification network

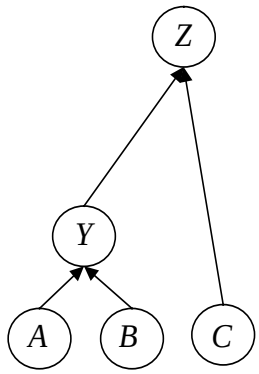
The work presented in the chapter so far has concentrated on the representation of the event recognition model. First it was shown how a classification network can be used to define a hierarchy of events and then two types of temporal constraint, `duration` and `precedes`, were incorporated in the event definitions. The notion of temporal distance has been introduced as a restricted type of `precedes` relation and it was shown how this constraint results in event definitions which contain precedence sequences of subevents. The resulting representation, incorporating all these features, is named a *temporal classification network* (TCN). This section examines the remaining aspects of the event recognition system, i.e., the database and the inference mechanism.

The assumptions that have been made in the previous sections about the representation of the event recognition model facilitate the design of a simple dynamic database for incremental event recognition. The input to the system is a stream of event occurrences, provided by the low-level event recognition system, e.g.

$$A(0,200), B(200,400), C(300,500), D(500,600), \ldots$$

Each occurrence is received by the temporal classifier at its end time point, e.g. $A(0,200)$ is received at 200, $B(200,400)$ at 400, etc., and it is added to the database. If more than one events are recognised at the same time, they are all added simultaneously to the database. When the database is updated, the relevant higher-order event recognition predicates are evaluated and any higher-order events that are recognised are added to the database. Event recognition is propagated through the TCN in that manner, until no more events can be recognised.

The database is indexed by the event signatures, allowing efficient retrieval of event occurrences. In fact, it can be structured as a list of stacks, each one corresponding to an event. The list is sorted using the partial support ordering of events in the classification network, which is also used by the inference algorithm. In this way the search for the required stack of occurrences is minimised. Moreover, each stack is automatically ordered in reverse recognition order, i.e., reverse order of end time points. The rationale for this ordering is that the most recent occurrences are more likely to be useful in event recognition than the older ones. Each stack is termed a *local node history*. Figure 3.9 illustrates this database structure for the simple DAG of Fig. 3.3.



A	(45,58)	(23,40)	(0,20)	
B	(47,55)	(25,30)		
C	(50,58)	(26,34)	(5,15)	
Y	(23,55)	(0,30)		
Z	(5,55)			

Figure 3.9: A simple classification network and the corresponding database of occurrences. The database is an indexed list of stacks, each of which contains a set of time stamps.

The size of the database is an additional design consideration. Local node histories can be restricted in size in two ways: by introducing an explicit maximum size and by calculating the earliest useful occurrence. The latter method is based on the observation that as time passes some event occurrences will not be usable in the recognition of further occurrences. In Fig. 3.9, for example, it may be the case that $A(0, 20)$ cannot be used in the recognition of further occurrences of Y, after time point 100. This decision is based on the temporal constraints in the definition of Y.

Using an event recognition model represented by a TCN and a dynamic database structured as described above, a simple inference algorithm can be designed to perform event recognition. The algorithm receives 0-order event occurrences and checks if they appear as terminal subevents in 1-order event definitions. If they do, it tries to recognise the corresponding 1-order events, using the local node histories of the other subevents in the definition. In the same way, event recognition is propagated through the TCN, using the partial event support order. Each recognised event occurrence is used to update the corresponding local node history. Algorithm 3.1 presents a narrative description of this process for conjunctive events, without rejecting subevents. The same algorithm can be used for disjunctive events, while the extension for repeating events and conjunctive events with rejecting subevents is simple. A detailed pseudocode description is provided in appendix A.

Input: 0-order event occurrences.

Output: recognised output events.

Globals: event recognition model; list E of events of order > 0, ordered according to the partial support order; the database.

TCN :

```
Update the 0-order node histories.
Initialise the list F with the new 0-order occurrences.
FOR EACH event e ∈ E DO:
    FOR EACH occurrence of the terminal subevent t of e, t(x,y) ∈ F DO:
        IF (y − x) satisfies the duration for t in the definition of e THEN:
            B_e ← evaluate-definition^a(e, t(x,y)).
            Add the new occurrences, B_e, of e to F.
            Update the node history for e.
```

[a] **evaluate-definition** builds all legal subevent sequences for the event e. It starts from the terminal subevent $t(x, y)$ and moves backwards through the sequence of supporting subevents.

Algorithm 3.1: Sketch of the inference algorithm for conjunctive events without rejecting subevents.

The worst-case complexity of the above algorithm is low polynomial in the three dimensions involved. To illustrate this point assume the following scenario:[9]

- The algorithm receives a single 0-order occurrence, e.g. $B(47, 55)$ from Fig. 3.9.
- If this event does not appear as a terminal subevent, nothing happens.
- Otherwise, the precedence sequence of the supported event, e.g. Y is evaluated. The worst-case complexity of this operation, i.e., the complexity of the **evaluate-definition** algorithm is $O(nm^2)$, where n is the number of subevents in the sequence and m the branching factor of the search tree for legal subevent sequences. The parameter m is a measure of the number of occurrences in a local node history that satisfy the temporal constraints. The quadratic complexity on m is caused by the evaluation of the distance constraints as the **evaluate-definition** algorithm moves backwards through the sequence of subevents. At each step backwards, the distance between the occurrences of the preceding subevent, e.g. A, and the preceded one, e.g. B, needs to be calculated. In practice the branching factor of this search is expected to be very small, i.e., a small number of occurrence sequences satisfy the constraints in the definition. In the example case, there is just one sequence $(A(23, 40), B(47, 55))$

[9] The complexity results presented here are independent of the example. The example is used simply for illustration.

satisfying the constraints and leading to the recognition of $Y(23, 55)$. The low complexity of this operation is a result of the precedence sequence assumption. Moreover, the use of the structured database should lead to further computational gains in the average case.

- If the recognised event is not used as a terminal subevent in other event definitions the process stops.
- Otherwise, the same process is repeated. In the example case, just once more, leading to the recognition of $Z(5, 55)$, with the sequence $(C(5, 15), E(23, 55))$. The number of times this can happen is restricted by the partial support order and it is of order $O(e)$, where e is the number of events in the model. This worst case will only be caused by a highly connected network, which is unlikely to be used in an event recognition system.

Thus, each time a low-level event, that is used as a terminal subevent in the TCN, is recognised the number of operations that need to be performed in the worst case is $O(nm^2e)$. The computational efficiency of the inference algorithm is a result of the two representational assumptions, i.e., the partial support order and the precedence sequence assumption.

3.6 Related work

In chapter 2 some of the popular approaches to event recognition were presented and the reasons why they cannot be applied to the task examined in this thesis were explained. In addition to these approaches, there are a small number of event recognition systems which are closer to the method proposed here and have been applied to similar types of problem. These systems have only briefly been mentioned in chapter 2. This section examines them in more detail, explaining their differences from the temporal classification network.

3.6.1 Temporal event recognition

Like the TCN, temporal event recognition systems use declarative models for recognition. Such an event recognition model defines events by associating temporally a set of subevents. Temporal association is achieved by a set of temporal constraints which need to be satisfied for the event to be recognised. A distinction is made between low-level (or *primitive*) events and

high-level (or *derived*) events. The former are the ones appearing in the input stream to the system.

As mentioned in section 3.3.3, primitive events are in essence instantaneous. This is the first difference of the TCN from those systems. The basic assumption in temporal event recognition systems is that the start and end points of durative events can be recognised individually, as instantaneous events. This is considered important, because on-line and predictive recognition is desired. In other words, as soon as the start of an event is recognised all of the events which are supported by it are considered *partially recognised*. For instance, as soon as the start of $A(0, 20)$ in Fig. 3.9 is recognised, $Y(0, ?)$ is expected to be recognised.[10] In order for $Y(0, 30)$ to be recognised, the end of $A(0, 20)$ and both the start and end of $B(25, 30)$ need to be recognised. In addition, all of the temporal constraints need to be satisfied. If any of these conditions are not satisfied, the partial occurrence $Y(0, ?)$ is not completed. In this example, if B was not recognised $Y(0, ?)$ would stay partially recognised for the remainder of the recognition session. In order to avoid that, the concept of '*window of relevance*' [26] has been introduced, which needs to be defined explicitly by the user of the system. At every time point, all partially recognised events are checked and if their window of relevance is exceeded they are removed.

There are some interesting problems with this *forward recognition* method:

- Multiple partially recognised occurrences of an event can exist at each point in time. For instance, if no B event occurred between the recognition of $A(0, 20)$ and $A(23, 40)$, two partially recognised occurrences of Y, $Y(0, ?)$ and $Y(23, ?)$, should exist in the system.
- Even when a partially recognised event is fully recognised, the partially recognised occurrence should be kept in case an alternative completion is found. For example, after the recognition of $A(0, 20)$ and $B(25, 30)$, $Y(0, 30)$ is recognised fully. However, when $B(47, 55)$ is recognised the occurrence $Y(0, 55)$ may also be a valid one. Therefore, $Y(0, ?)$ should not be removed after the recognition of $Y(0, 30)$.
- A partially recognised event can cause the partial recognition of a higher-order event. For instance, as soon as $Y(0, ?)$ is created, $Z(0, ?)$ needs to be created too. Also whenever, the partial occurrence $Y(0, ?)$ is updated, due to the recognition of further

[10] The '?' end point signifies the fact that the occurrence is only partially recognised.

events, e.g. the detection of the end of $A(0, 20)$, $Z(0, ?)$ should be updated, too.

These problems have a serious effect on the space and computational complexity of the forward recognition algorithm. However, this cost may be acceptable if partially recognised events are useful to the system, e.g. for the purpose of focus of attention.

In contrast to this approach, *backward evaluation* of event definitions has been chosen for the TCN. This choice was motivated by the different type of event used in the system. Namely, the assumption was made that an event cannot be recognised before it ends. For this type of event it seems more appropriate to wait until the terminal subevent in an event definition is recognised and then check whether the event can be recognised. The process taking place during the recognition of events that are not used as terminal subevents stores the event occurrences in a way that allows efficient retrieval by the backward evaluation algorithm. Limited use of forward matching can be made by constructing partial subevent sequences for an event and checking the temporal constraints. This can only be done locally though, i.e., for one event definition, as partially recognised subevents are not of use in the recognition of higher-order events.

Intuition suggests that forward recognition is more appropriate for on-line recognition systems than backward recognition. The problems described above make this claim less obvious. The backward recognition approach is simpler, as it does not require the use of windows of relevance and the continuous updating of partially recognised events. It is also more space efficient, as no partial occurrences need to be stored. In terms of computational complexity, backward recognition is unlikely, in average, to be more efficient than forward recognition. However, there are cases where this is also possible. As a simple example, assume the definition of an event Z in terms of the subevents A, B, C, D. Assume also the following sequence of recognised event occurrences:

$$A(0, 2), A(2, 3), B(3, 4), C(3, 6), B(5, 6), C(5, 8), C(8, 10).$$

Given that the window of relevance is large enough, a large number of partially recognised events for Z will be constructed:

10 sequences of (A, B, C),

4 sequences of (A, B),

2 sequences of (A).

If D is now recognised, the forward recognition algorithm will have to examine at least the 10

(A, B, C) sequences to check whether Z can be recognised. In the worst case, the backward recognition algorithm would need to generate all 10 (A, B, C) sequences, which would cost 15, instead of 10 operations. However, if just one C occurrence did not satisfy the distance constraint between C and D, this number would be reduced. For instance, if $C(8, 10)$ was too close to the recognised D, only 9 operations would be needed for the backward matching algorithm.

The above example illustrates a problem, which has been also observed in [25], namely, that the highest computational cost of the forward recognition algorithm is incurred by the multiple copies of partially recognised events. This problem becomes more serious with the use of a high-order classification network, where the number of partially recognised events can increase significantly. For this reason, a more thorough comparison of the two approaches is required, in order to determine the conditions under which each one should be preferred.

Another difference of the TCN from the temporal event recognition systems is the range of allowed temporal relations in the representation. These systems use more expressive temporal logics than the TCN, which allow a variety of temporal relations between subevents in an event definition. Usually, all temporal relations are interpreted into the three basic point-based relations, i.e., $<, =, >$, in order to make the temporally reasoning task tractable.

The TCN facilitates a more limited range of temporal relations, i.e., stronger assumptions about the order of subevents in an event definition are made. The most restrictive assumption is the precedence sequence, which, however, results in a substantial gain in the computational complexity of the recognition algorithm. If this assumption is removed, the evaluation of event definitions will become more complex and a constraint propagation algorithm will be needed for that purpose. Examples of such algorithms appear in [26] and [46].

In addition to the above technical details, there is a difference of emphasis between the TCN and other temporal event recognition systems. The emphasis in the latter is in the expressiveness of the event definitions and the efficiency of the local constraint propagation algorithms. These are undoubtedly very important issues in an event recognition system. However, equally important is the global structure of the system, which has been the focus point in the design of the TCN. A graphical representation has been proposed for that purpose, which imposes a hierarchical ordering of events in the recognition model. Furthermore, the TCN has been viewed as a computational network and different types of event recognition functions have

been considered. This issue has not been examined in other temporal event recognition systems. This alternative view of event recognition is the main contribution of the work presented in this chapter. The TCN representation provides an efficient approach to event recognition and a suitable framework for the development of the parameter refinement methods presented in the following chapters.

3.6.2 An alternative view of temporal classification

The basic structure of a TCN event recognition model can be represented by a *context-free grammar* (CFG), in which 0-order events are the terminal symbols and higher-order events non-terminal.[11] For example, the DAG of Fig. 3.3 can be represented as follows:

```
<Z> ::= C,<Y>
<Y> ::= A,B
```

In fact, the full expressive power of a CFG is not necessary, because the network can be flattened into an equivalent set of sequences of 0-order events. in the above example, there is just one such sequence, i.e., (C, A, B). More than one sequence is generated if disjunction is used in the TCN and/or there is more than one output node. The resulting set of sequences corresponds to all possible combinations of 0-order events that can lead to the recognition of output events. Such flat sequences can be represented by *finite-state* or *regular* grammars, or their equivalent *finite-state automata* (FSA). The FSA for the above simple example is shown in Fig. 3.10.

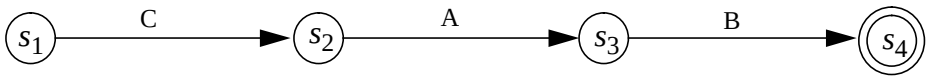


Figure 3.10: A finite state automaton for a simple CFG.

The focus of the grammatical representation is on the temporal succession of events, rather than the temporal relations between them. For example, the idea of selective use of event occurrences, from a sequence, in the recognition of an event is not a natural one for a grammar-based inference system. For similar reasons, the treatment of overlapping events is problematic. However, both these problems can be solved by augmenting the grammar with the appropriate

[11] A textbook introduction to grammars can be found in [42].

temporal relations. A more difficult problem is the representation of rejecting events, as negation in grammatical inference is defined by failure of match, i.e., illegality of a sequence. If the grammar was also augmented to include "negative symbols", the end result would be equivalent to the rules of the production system, by which the discussion in this chapter started.

The correspondence of a TCN to a grammar provides a different insight to the representation of repeating events in a TCN model. One could view these events as a restricted type of loop in the DAG, where the repeating event supports itself. This alternative representation would suggest a recursive inference mechanism for repeating events, rather than the iterative one used in the proposed algorithm. The reason for not following this path is that it introduces disturbing exceptions to the partial support order, which is useful for providing efficient classification. An additional problem is that counting is not a natural property of grammars and for this reason the representation of the bounds on the number of repetitions presents a difficulty.

Grammatical representation has been used in syntactic pattern recognition systems, an example of which is the speech recognition system, HARPY [53]. HARPY is a system that identifies legal sentences in continuous speech. It operates directly on the raw signal and for this reason it incorporates several processing elements. The one which is of relevance to the work presented here is the *knowledge network*, which performs the mapping of phoneme sequences to legal sentences. The knowledge network is constructed using four different types of knowledge:

- Legal sentences are described using a finite-state grammar, which provides a hierarchical decomposition of the sentences into words. The following is an example appearing in [53]:

  ```
  <SENT> ::= [ <SS> ]
  <SS>   ::= please help <M> | please show <M> <Q>
  <Q>    ::= everything | something
  <M>    ::= me | us
  ```

  where '[', ']' denote the beginning and end of the utterance.
- Words are described in terms of phonemes in a pronunciation dictionary, which contains alternative pronunciations of a word. The words in the above example are described as follows:

  ```
  everything (-,0) (EH,EH2) V R IY2 TH IH3 NX
  help       (-,0) HH AA3 EL3 (← (-,0),-) P
  me         (-,0) M IY
  ```

```
please    (← (-,0),-) (P (L,LL2), PL (L,0)) IY (Z{4},(Z,0) S)
show      (-,0) SH AA5 (OW,0)
something (-,0) S AA M TH IH3 NX
us        (-,0) IH6 S (HH,0)
[         -
]         -
```

where '-' denotes silence, (x,0) means that x may be missing and $\{n\}$ is an exceptional duration constraint on a phoneme.

- A set of juncture rules determining the transition between words.
- A set of templates for phoneme recognition, which also provide typical duration ranges.

All this knowledge is processed by a knowledge compiler to construct a DAG, the vertices of which correspond to phonemes, including silence, and the edges to legal transitions. Duration constraints are imposed on the phonemes.

The knowledge network is used to provide stepwise explanation of the received signal, which is sampled at regular intervals. The duration of phonemes is calculated by allowing a delay in each node, restricted by the duration constraints. The system outputs one event, i.e., one sentence, which is considered the best match to the utterance, according to some distance metric. The match between the utterance and the model is updated incrementally, while moving through the knowledge network.

As mentioned also in chapter 2, HARPY was the only system which satisfied the requirements of the ARPA speech understanding project. However, it suffered form one important problem: the size of its knowledge network. This is a result of the compilation of high-level knowledge, i.e., sentence specification, to low-level phoneme transitions. In comparison, the TCN provides a more compact representation, by preserving the hierarchical definition of events. Other differences of the two systems are:

- HARPY does not deal with overlapping events and rejecting subevents, for the reasons explained above about grammatical representations.
- It also cannot deal with repeating events, which is a further restriction to the grammar.
- TCN outputs all matching sentences, or none if none satisfies all constraints. HARPY uses a heuristic distance metric instead, to decide on the best-matching sentence.

3.7 Summary and critique

A particular type of event recognition has been examined and a suitable representation for event recognition models has been presented. The temporal classification network allows the representation of hierarchical event definitions and the incorporation of temporal constraints. The proposed representation allows the definition of three types of event, i.e., conjunctive, disjunctive and repeating, and the use of negative evidence in event definitions. It also permits two types of temporal constraint, the unary `duration` and the binary `precedes`, which operate on intervals associated with event occurrences. Two important assumptions about the structure of the model have been made, i.e., partial support order and precedence sequence, allowing the design of a simple and efficient inference algorithm for event recognition.

The main restriction of the TCN is in terms of the expressive power of the representation. The system could be extended to relax any of the following assumptions:

- Events do not need to be defined hierarchically. For example, recursive event definitions could be permitted. Such an extension would introduce loops in the network, which would complicate the inference mechanism. However, restricted use of loops might be manageable.
- The use of non-boolean functions could be extended to allow probabilistic evidence combination. This extension would allow event recognition under model or data uncertainty.
- Further types of temporal relation between subevents could be permitted. This would allow more flexible definition of events, which might be necessary for some applications. For example, the terminal subevent in an event definition may not be known with certainty. In that case a less restrictive relation between the potential terminal subevents, than the `precedes` relation may be necessary.
- The strict precedence sequence assumption could be relaxed to allow a larger number of `precedes` relations. Again this extension is associated with higher computational cost.
- The temporal network could be integrated with a non-temporal one. An example of this is the finite-state grammar used for the definition of sentences in the HARPY system. In such a system the quantitative temporal relations between words may not be known. In the TCN representation, such relations can be handled, using constraints

with undefined range limits.

In addition to the extension of the representation scheme, the inference algorithm could be modified to perform a selection of the most likely sequence of recognised events. The philosophy of the proposed system is closer, on that issue, to that of temporal event recognition systems, where all valid inferences are taken to be true. This seems a more appropriate approach for a system which does not deal with purely sequential input, but allows overlapping events at different levels of the hierarchy.

Part III

Refinement of Temporal Parameters

Chapter 4

Incremental parameter refinement

Accurate setting of the temporal parameters of an event recognition model can be a tedious and difficult task. This chapter examines the refinement of an initially inaccurate model using training data. The event recognition model is assumed to be represented as a temporal classification network, containing numeric duration and distance parameters. The training data is presented as two separate streams of low and high-level occurrences. Refinement aims at the correct recognition of high-level events in the sequence of low-level occurrences. The format of the data as two separate streams introduces the problem of establishing the correspondence between the two streams and building a set of training examples. The concept of *event support trees* is presented as a solution to this problem. Furthermore an incremental refinement method is proposed, which guarantees the optimal solution, minimising a given cost function.

4.1 Refinement of a temporal classification network

This chapter presents a knowledge refinement approach to the task of automatically setting the temporal parameters of an event recognition model. The focus of the knowledge refinement enterprise is on the modification of a symbolic model, with the use of a small training set.[1] The underlying assumption is that the model, in its initial state, achieves a low performance on the modelled task, e.g. classification. The refinement algorithm is required to extract information from training data, drawn from the modelled process, and use it to improve the model. A reasonable assumption, in this context, is that the initial model is close to the desired one and therefore only small changes need to be performed. This is a necessary assumption, imposed by the restrictive size of the training data. Purely empirical learning methods, e.g. MLPs, rule

[1] One could extend this perspective to include subsymbolic systems, but that would be misleading in the context of this thesis.

induction, decision tree induction, etc., are not applicable to knowledge refinement, due to the scarceness of empirical data and the complexity of the model. However, empirical learning methods and knowledge refinement share many of the basic ideas of inductive inference, since they both extract information from training data.

Section 2.2, in chapter 2, has presented briefly the most influential approaches to knowledge refinement. The common feature of these methods which makes them inappropriate for the task examined here, is the assumption of a clear mapping of input to output in the training data. The training examples are assumed to consist of a set of input values, which describe an object, and an output value, which provides the desired classification of the object. This assumption is invalid for time-dependent data, such as an event sequence, where the refinement method needs to establish the correspondence between the low-level event occurrences and the desired high-level ones. Additionally, in the case where more than one sequence of low-level occurrences can cause the recognition of a high-level one, a mechanism is needed for choosing among alternative low-level sequences. The methods presented in this and the following two chapters utilise the structure of the temporal classification network (TCN), to provide solutions to this problem.

The refinement of a TCN model can take various forms, corresponding to tasks of different complexity. Two naturally distinct refinement tasks are the adjustment of numerical parameters and the restructuring of the model. The former task involves the modification of the temporal constraints in the TCN model, i.e., the distance and duration ranges, which determine legal precedence sequences. Model restructuring in a TCN involves the deletion or insertion of links and nodes in the network, i.e., the removal or addition of event definitions and support dependences. This thesis concentrates on the refinement of numerical parameters, which usually involve a higher degree of uncertainty. It seems intuitive that a human expert, designing the event recognition model, would be less certain about the correct setting of the temporal constraints than the dependence between events.

Based on the properties of a TCN model, the task of parameter refinement can be further subdivided into subtasks, as shown in Fig. 4.1. The first important aspect of the task is the amount of available supervision. In this chapter and chapter 5 it is assumed that training feedback is provided for *all* of the events in the TCN. This type of supervision is termed *full*, as opposed to *partial supervision*, which assumes training information only for a subset of the

TCN nodes and is examined in chapter 6. The two chapters which deal with refinement under full supervision present two different approaches, each with its own merits and limitations. The approach presented in this chapter is an incremental one, with low memory requirements, achieving locally optimal refinement. The one presented in chapter 5, uses an extended memory scheme which accumulates information over time. This approach allows independence of the refinement algorithm from the order of presentation of the data, but is space inefficient. Moreover, refinement is performed with the help of a heuristic search method, which provides tolerance to noise in the data, but does not guarantee optimality.

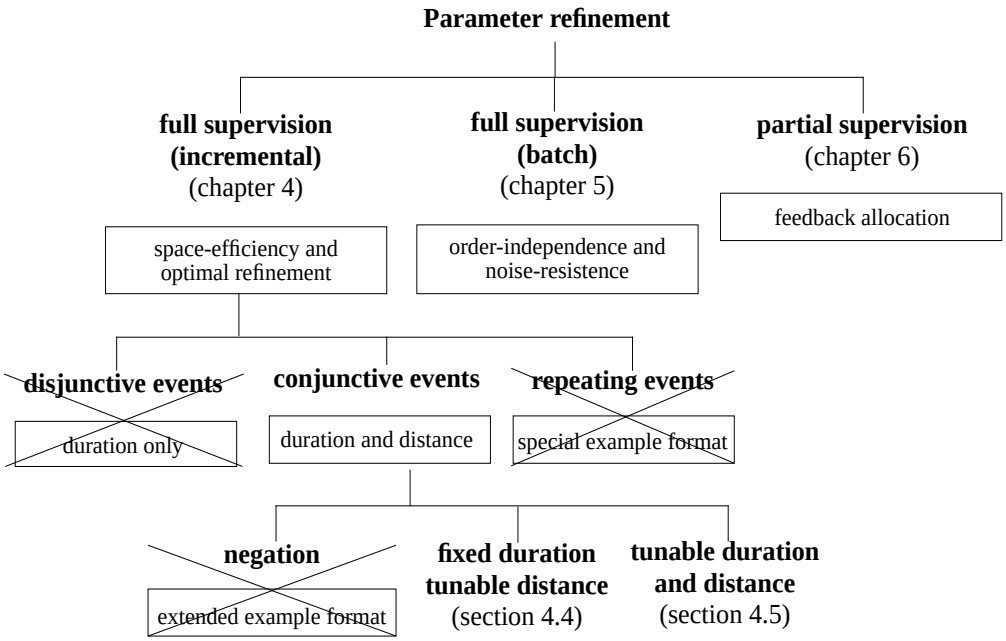


Figure 4.1: Parameter refinement tasks in a TCN. The task examined in this chapter is subdivided into subtasks. Interesting features of each task are included in the boxes underneath them. Crossed-out items are not examined.

Additionally, the following assumptions are made in this chapter:

- Only conjunctive event definitions are examined. The extension to disjunctive events is trivial, but repeating events cause further problems as will be seen in chapter 5.
- Only one terminal subevent per definition is assumed. As mentioned in chapter 3, the case of temporally independent precedence sequences is considered a degenerate one.

- No rejecting subevents are allowed in event definitions.

Further to these assumptions, all the examples in the discussion that follows do not make multiple use of subevents in a definition. This is not an assumption in any of the refinement algorithms and aims only to simplify the explanation.

4.2 Properties of the search space

Learning and refinement methods perform a search in the space of alternative models. The goal of the search is a model that accounts best for the training data. The search space for the refinement of a TCN model is defined by its temporal parameters, which are the ones being refined. This section examines some interesting properties of this space.

The refinement methods described in this chapter deal with simple conjunctive events, without rejecting subevents. For example, an event Z may be defined in terms of the subevents (A, B, C, D) as follows:

`IF` $A(i_A)$ `AND duration`(Z, A, d_A)
`AND` $B(i_B)$ `AND duration`(Z, B, d_B) `AND precedes`(Z, A, B, s_{AB})
`AND` $C(i_C)$ `AND duration`(Z, C, d_C) `AND precedes`(Z, B, C, s_{BC})
`AND` $D(i_D)$ `AND duration`(Z, D, d_D) `AND precedes`(Z, C, D, s_{CD})
`THEN` $Z(i_Z)$, `WHERE` $i_Z^- = \min(\{i_A^-, i_B^-, i_C^-, i_D^-\})$ `AND` $i_Z^+ = i_D^+$,

where $i_x = (i_x^-, i_x^+)$ is the interval associated with the event x and `duration` and `precedes` the elastic temporal constraint predicates, as defined in chapter 3. To avoid confusion, examples in this chapter use the first letters of the alphabet to denote subevents and the last letters to denote the defined events. Figure 4.2 provides the precedence net for the above example.

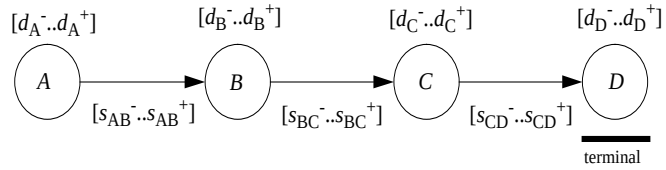


Figure 4.2: A simple conjunctive precedence net. The duration constraints appear above of the subevent nodes and distance ranges below the links between subevents.

The proposed parameter refinement method is an incremental one. The model is refined

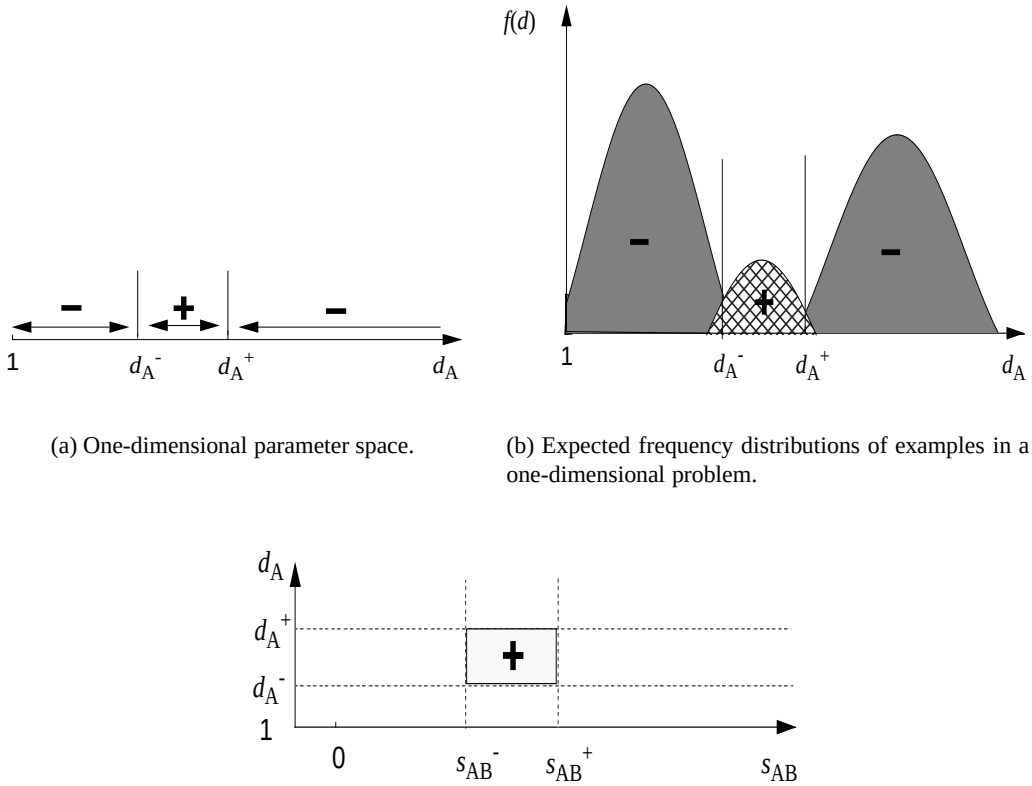
every time new information is received, i.e., every time a new event is recognised. More specifically, whenever an occurrence of the terminal subevent, D in the above example, is recognised, the refinement algorithm updates the temporal parameters of the corresponding definition, aiming to achieve the following two goals:

a. Prevent the recognition of event occurrences which could be recognised and should not have been (*type 1 error*). This is achieved by *contracting* the parameter ranges so that they are no longer satisfied by the corresponding sequence of subevent occurrences. When this is achieved, the model is said to *exclude* the negative example.

b. Ensure the recognition of occurrences which would not be recognised, while they should have been (*type 2 error*). This is achieved by *expanding* the parameter ranges, so that they are satisfied by the sequence. After this change, the model *covers* the positive example.

The training feedback provides information about the event occurrences that should be recognised and this information is assumed to be complete.

In the example event definition there are 14 parameters, 6 distance and 8 duration range limits. More generally, a conjunctive event definition of this type with n subevents will have $4n - 2$ parameters. Each parameter range separates the set of all possible parameter values into three mutually exclusive subsets. This is illustrated, by the three adjacent regions on the duration axis for subevent A in Fig. 4.3(a). The two outer regions correspond to duration values that do not satisfy the constraint and should contain the majority of observed durations which do not cause Z to be recognised (*negative examples*). Similarly the inner region should include most of the *positive examples* of Z. If this duration range was the only constraint in the definition, the expected distribution of positive and negative examples might be as in Fig. 4.3(b). This graph shows that in reality the three regions are not completely mutually exclusive, due to noise in the data. In a problem of higher dimensionality, the inner region may also include some negative examples, which do not satisfy one of the remaining constraints in the definition. This is shown in the two-dimensional parameter space in Fig. 4.3(c). Positive examples in two dimensions are the ones inside the rectangular region, i.e., these which satisfy both constraints. Similarly, the complete parameter space for the definition of event Z above has 7 dimensions, one for each parameter range. The positive area in this space is represented by the hyper-rectangle, defined by the 7 ranges.



(a) One-dimensional parameter space.

(b) Expected frequency distributions of examples in a one-dimensional problem.

(c) Two-dimensional parameter space.

Figure 4.3: The parameter space for a conjunctive event definition.

Therefore, if Z was only supported by A, the task of the refinement algorithm would be to search for the values of d_A^- and d_A^+ which would keep each of the three regions in Fig. 4.3(a) as *pure* as possible, i.e., containing only the correct type of examples. In a longer definition sequence, the search becomes more complex, as a larger set of duration and distance constraints have to be satisfied for Z to be recognised. The refinement algorithm needs to examine possible combinations of changes to the parameter settings which lead to the desired result. A further assumption which is made in this chapter is that a purely positive region exists in the n-dimensional space, where n the number of parameter ranges. This region includes all positive and excludes all negative examples. This is an unrealistic assumption which will be relaxed in chapter 5.

One of the most important issues in the task of TCN parameter refinement is the construction of the training set from the input and feedback data. Assume for example, that the following sequence of subevents is received as input:

$$A(1,3), A(2,4), A(3,5), B(2,5), B(5,7), C(7,9), C(9,10), D(11,12)$$

and event occurrence $Z(2,12)$ should be recognised as a result. There are 6 possible ways of achieving the recognition of $Z(2,12)$, namely with the use of the following 6 subevent sequences:

$$\begin{aligned}
A(2,4), B(2,5), C(7,9), D(11,12) &\rightarrow Z(2,12),\\
A(3,5), B(2,5), C(7,9), D(11,12) &\rightarrow Z(2,12),\\
A(2,4), B(5,7), C(7,9), D(11,12) &\rightarrow Z(2,12),\\
A(2,4), B(2,5), C(9,10), D(11,12) &\rightarrow Z(2,12),\\
A(3,5), B(2,5), C(9,10), D(11,12) &\rightarrow Z(2,12),\\
A(2,4), B(5,7), C(9,10), D(11,12) &\rightarrow Z(2,12).
\end{aligned}$$

The coverage of each of these sequences may require a different modification of the parameters than the others. However, only one sequence needs to be covered. For this reason the 6 sequences are named *alternative positive examples*. All other possible combinations of subevent occurrences are negative examples and there are 6 such sequences:

$$\begin{aligned}
A(1,3), B(2,5), C(7,9), D(11,12) &\nrightarrow Z(2,12),\\
A(3,5), B(5,7), C(7,9), D(11,12) &\nrightarrow Z(2,12),\\
A(1,3), B(5,7), C(7,9), D(11,12) &\nrightarrow Z(2,12),\\
A(1,3), B(2,5), C(9,10), D(11,12) &\nrightarrow Z(2,12),\\
A(3,5), B(5,7), C(9,10), D(11,12) &\nrightarrow Z(2,12),\\
A(1,3), B(5,7), C(9,10), D(11,12) &\nrightarrow Z(2,12).
\end{aligned}$$

Each of these sequences needs to be excluded by the model. Similarly, if no occurrence of Z was required, all 12 sequences would be negative examples of Z.

One common subevent occurrence in all of these sequences is the terminal one, i.e., $D(11,12)$, which triggers the example construction process. This observation has led to the

concept of the *event support tree* (EST), which is instrumental in the design of the refinement algorithms in this and the following two chapters. An EST is considered to be one example, either positive or negative, of an event and has the following properties:

- The root of the tree corresponds to the occurrence of the terminal subevent.
- Each node of the tree represents a subevent occurrence.
- Each horizontal level of the tree contains all occurrences of a subevent.
- The children of each node correspond to all occurrences of the preceding subevent.
- Each path from the root to the leaf of the tree is a complete sequence of subevents.
- All leaves of the tree are equidistant from the root.

Figure 4.4 shows the EST for the positive example of $Z(2, 12)$ above. The negative EST is constructed similarly.

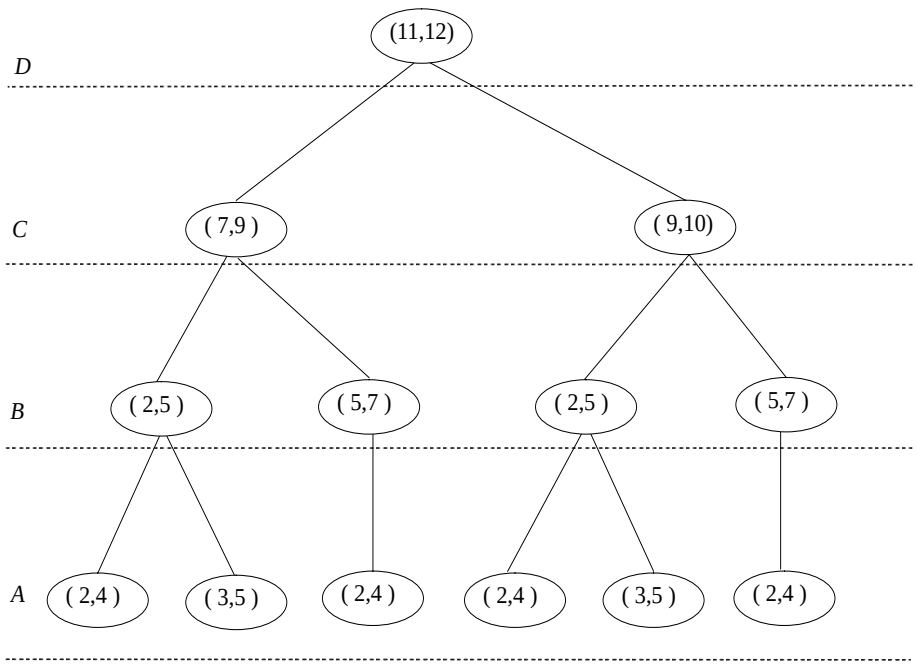


Figure 4.4: An example of an event support tree (EST).

The event support tree is a compact representation which captures the concept of alternative example sequences. It achieves that by utilising the special status of terminal subevents in TCN event definitions. Furthermore, the structure of the EST is used by all of the refinement

algorithms in the thesis, to guide the search for new parameters. A more compact variant of the EST is the *event support network* (ESN), which avoids duplication of subevent occurrences by relaxing the assumption of a tree structure. Figure 4.5 shows the (ESN) for the EST of Fig. 4.4. The reason for choosing the EST, rather than the ESN representation is that it simplifies the design of the refinement algorithms. However, the compactness of the ESN is an attractive feature and is examined again in chapter 5.

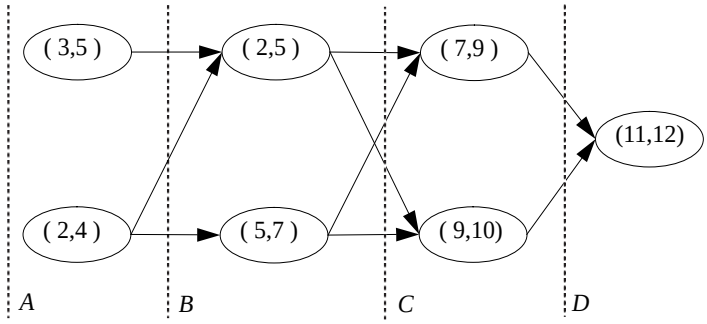


Figure 4.5: An example of an event support network (ESN).

4.3 Restricted-memory parameter refinement

In an incremental learning or refinement method the memory structure, in which information from the training data is accumulated, plays an important role. The method presented in this chapter uses a restricted memory structure for the refinement of the parameters in a TCN.

The usual approach to the representation of the training data in empirical learning is to view instances as points in the n-dimensional parameter space. This representation is also known as the *instance space*. According to this approach, the six alternative positive sequences of the above example would correspond to the following ones:

$$
\begin{array}{llllllll}
d_A = 2, & s_{AB} = -2, & d_B = 3, & s_{BC} = 2, & d_C = 2, & s_{CD} = 2, & d_D = 1, \\
d_A = 2, & s_{AB} = -3, & d_B = 3, & s_{BC} = 2, & d_C = 2, & s_{CD} = 2, & d_D = 1, \\
d_A = 2, & s_{AB} = 1, & d_B = 2, & s_{BC} = 0, & d_C = 2, & s_{CD} = 2, & d_D = 1, \\
d_A = 2, & s_{AB} = -2, & d_B = 3, & s_{BC} = 4, & d_C = 1, & s_{CD} = 1, & d_D = 1, \\
d_A = 2, & s_{AB} = -3, & d_B = 3, & s_{BC} = 4, & d_C = 1, & s_{CD} = 1, & d_D = 1, \\
d_A = 2, & s_{AB} = 1, & d_B = 2, & s_{BC} = 2, & d_C = 1, & s_{CD} = 1, & d_D = 1.
\end{array}
$$

Using this representation, the usual approach to symbolic learning, e.g. by ID3 [82], AQ15 [60], CN2 [16], etc., is to try to separate the n-dimensional space into mutually exclusive regions, each containing either negative or positive examples. In the examined refinement task, this approach suffers from the following problems:

- It cannot deal with alternative positives, i.e., each positive point in the instance space is treated individually. This would guide the refinement towards the coverage of all alternative positive sequences, which is not required and might be unattainable.
- The structure of the model needs to be taken into account. The 7-dimensional representation of the above instance space corresponds to a 14-parameter model, which has a particular conjunctive structure. The common empirical learning methods make specific assumptions about the structure of the model, e.g. decision tree by ID3, CNF by CN2 and AQ15. It is thus difficult to preserve the structure of the precedence sequence with these methods.
- The standard learning methods are purely empirical and need to be extended to incorporate a model-based bias.

The algorithm presented in the rest of this chapter performs knowledge refinement, rather than purely empirical learning. It has a bias for minimum model change, which is implemented by a cost function, measuring the degree of change imposed to the model by each proposed modification. At the same time it makes the simplifying assumption that there is no noise in the training data, i.e., there are no contradictions between the training examples. On that basis, the algorithm performs a search for the optimal parameter modification, according to the model-based bias. Although unrealistic for most applications, the no-noise assumption seems less unreasonable in the light of a small training set. It is relaxed in the algorithm presented in the following chapter, which incorporates more of the standard empirical learning ideas.

The philosophy of the optimal refinement algorithm (ORA), presented here, is based on the *version space* approach to learning, introduced by Tom Mitchell in the late '70s [64]. The version space of a learning problem is a memory structure which holds the information extracted from a sequence of examples in the form of two sets of models: the most general ones, which just exclude all negative examples and the most specific, which just cover all positive examples. Individual models in each set represent alternative solutions to the problem, i.e., alternative models which cover all positive and exclude all negative examples. Mitchell

[64] presents a simple learning algorithm, called *candidate elimination*, which uses the version space to perform an exhaustive search for all legal models corresponding to a set of examples. A model belonging to the most general set becomes illegal, when it becomes less general than one of the models in the most specific set and vice versa. In practice this exhaustive approach suffers from high storage demand. For this reason, ORA maintains a single pair of models, one most general and one most specific. These are selected with the use of the model-based cost heuristic each time a new example is considered. The effect of this simplification is that the algorithm becomes dependent on the order of presentation of the data, since it uses a very restricted memory structure. A compromise between the two approaches is the *beam search* method, used in [59], which maintains the m best models, where m is a user-specified parameter. ORA can easily be extended to exhibit a version-space or a beam-search behaviour.

Each of the two models in the memory of ORA consists of a sequence of temporal constraints. Thus, three pairs of ranges are maintained for each constraint:

a. The *most general generalisation (MGG)*, which is the most general model for the range.
b. The *least general generalisation (LGG)*, corresponding to the most specific model.
c. The *preferred parameter setting (PPS)*, which is the range that would be used if training stopped at that point. The PPS is an extension of version spaces to take into account the original model. Unless initially specified by the model, the PPS is the same as the MGG, i.e., there is a bias for positive classification.

Using the symbols $\{>_g, \geq_g, =_g, \leq_g, <_g\}$ to denote the relative generality of one distance range against another, the following hold for the MGG, PPS and LGG ranges of subevents A, B in the definition of Z above:

$$\begin{aligned} MGG_d(A) &\geq_g \quad PPS_d(A) \quad \geq_g LGG_d(A), \\ MGG_s(A,B) &\geq_g \quad PPS_s(A,B) \quad \geq_g LGG_s(A,B), \end{aligned}$$

where the index for each of the three ranges specifies whether the range is a duration, d, or a distance, s, constraint. Figure 4.6 illustrates the relationship between the three ranges on the duration d_A axis. The corresponding models for the whole definition of Z, are conjunctions of

the individual parameter models:

$$
\begin{aligned}
MGG(Z) &= MGG_d(A) \wedge MGG_s(A,B) \wedge MGG_d(B) \wedge MGG_s(B,C) \\
&\quad \wedge MGG_d(C) \wedge MGG_s(C,D) \wedge MGG_d(D), \\
LGG(Z) &= LGG_d(A) \wedge LGG_s(A,B) \wedge LGG_d(B) \wedge LGG_s(B,C) \\
&\quad \wedge LGG_d(C) \wedge LGG_s(C,D) \wedge LGG_d(D), \\
PPS(Z) &= PPS_d(A) \wedge PPS_s(A,B) \wedge PPS_d(B) \wedge PPS_s(B,C) \\
&\quad \wedge PPS_d(C) \wedge PPS_s(C,D) \wedge PPS_d(D).
\end{aligned}
$$

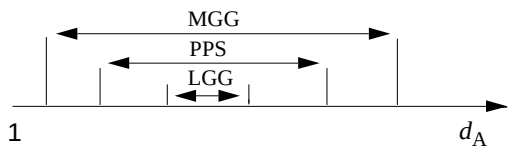


Figure 4.6: The LGG, MGG and PPS ranges for the duration of subevent A.

ORA incrementally updates the three models for an event definition, as new examples are received. This updating takes place every time a terminal subevent is recognised and the corresponding ESTs are constructed as described above. ESTs for positive examples, referred to as *positive* ESTs, are used to expand the $LGG(Z)$ model, if this does not cover at least one of the positive sequences. This expansion process is known as *generalisation* and is performed by the ORAgen algorithm. ORAgen calculates the minimum change to $LGG(Z)$, which is sufficient to cover the example. Similarly, negative examples may cause the contraction of the $MGG(Z)$ model, performed by the *specialisation* algorithm ORAspec. Note that due to the relation of the three models, the $LGG(Z)$ and $MGG(Z)$ models may need to be updated, although the $PPS(Z)$ model, containing the actual parameter settings, classifies correctly the positive and negative examples. The $PPS(Z)$ model will only need to be changed when the generality ordering is violated. In that case it is expanded or contracted sufficiently to lie between the $MGG(Z)$ and $LGG(Z)$ models. Thus, specialisation and generalisation affect primarily the memory structure and not the actual model. Refinement is achieved as a side-effect of this process.

Algorithm 4.1 gives an overview of the ORA parameter refinement algorithm. The generalisation and specialisation algorithms are examined in detail in the following sections. Section

4.4 deals with the case where the duration parameters are fixed and only the distance parameters are refinable. This assumption reduces the size of the search space and the complexity of the generalisation and specialisation algorithms. Section 4.5 then relaxes this assumption and presents the final versions of the two algorithms.

Input: The definition of an event e; a new positive or negative example x for e, in the EST format; the three models: $MGG(e), LGG(e), PPS(e)$.
Output: The new set of models for e: $MGG(e), LGG(e), PPS(e)$.
ORA$(e, x, MGG(e), LGG(e), PPS(e))$ **:**
`IF` x is positive:
    $LGG(e) \leftarrow$ **ORAgen**$(LGG(e), x)$
    `IF` $LGG(e) >_g PPS(e)$: $PPS(e) \leftarrow$ **expand-PPS**$(PPS(e), LGG(e))$
`ELSE`:
    $MGG(e) \leftarrow$ **ORAspec**$(MGG(e), x)$
    `IF` $MGG(e) <_g PPS(e)$: $PPS(e) \leftarrow$ **retract-PPS**$(PPS(e), MGG(e))$
`RETURN` $(MGG(e), LGG(e), PPS(e))$

Algorithm 4.1: The ORA parameter refinement algorithm.

4.4 Refining the temporal distance parameters

4.4.1 Distance-only generalisation

Under the assumption that duration ranges are fixed, the definition of event Z above contains three pairs of refinable distance ranges. Each of these ranges is associated with an MGG_s, an LGG_s and a PSS_s range. The aim of the generalisation algorithm is to modify the three LGG_s ranges, in response to positive examples.

In order to simplify the problem further, assume that Z is defined as a sequence of just two subevents A and B, as follows:

`IF` $A(i_A)$ `AND` $B(i_B)$ `AND` `precedes`(Z, A, B, s_{AB})
`THEN` $Z(i_Z)$, `WHERE` $i_Z^- = \min(\{i_A^-, i_B^-\})$ `AND` $i_E^+ = i_B^+$,

ignoring the fixed duration constraints which are assumed to be satisfied always. In that situation, the only refinable range is s_{AB}. Assume also that $LGG_s(A, B) = [-1..1]$ and the following sequence of subevent occurrences are recognised:

$$A(4, 5), B(7, 9), A(10, 11), B(9, 11),$$

in which $Z(4, 9)$ and $Z(9, 11)$ should be recognised. The distance between the first two occurrences of A and B is $(7 - 5) = 2$ and the distance between the second pair of occurrences is $(9-11) = -2$. In this case, the refined range is $LGG_s(A, B) = [-2..2]$, i.e., the $LGG_s(A, B)$ range has been extended by a unit each way. Figure 4.7 illustrates the effect of the two generalisation steps. With this schematic illustration in mind, the decrease of the lower bound of the $LGG_s(A, B)$ range is named *left generalisation* and the increase of the upper bound *right generalisation*.

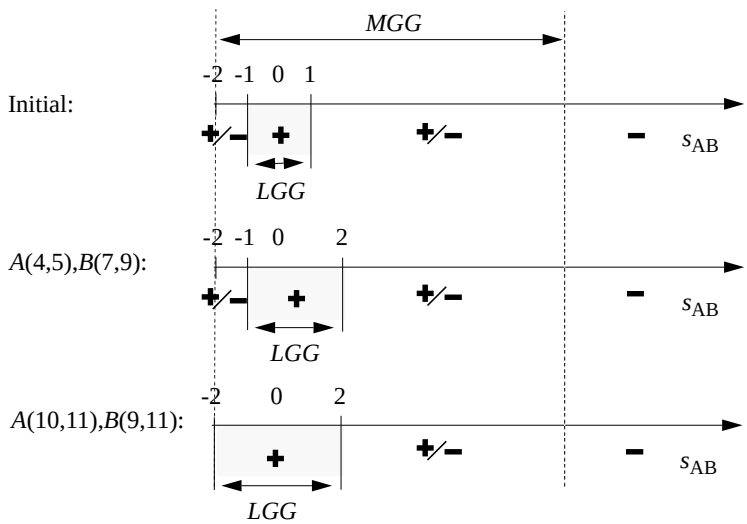


Figure 4.7: One-dimensional incremental generalisation. +: positive range; -: negative range; +/-: region of uncertainty. It is also assumed that the maximum duration of B is 2, which determines the minimum distance between A and B to be -2.

In the simple two-subevent definition, the refinement decision is clear. Extending, however, the definition to the original four-subevent one, more than one alternative positive examples may exist and a cost function is needed to decide on the best choice. The cost function that is used in the ORAgen generalisation algorithm is called *generalisation cost* and is defined as the degree of change imposed by a positive example to the original LGG_s model. In the simple case examined above, each of the two positive examples causes the minimum change of one unit to the LGG_s range. Given a distance value s_{hj}, between two subevent occurrences and

the range $LGG_s(e_h, e_j) = [s^- .. s^+]$, the generalisation cost function is defined as:

$$c_g(s_{hj}) = \begin{cases} s^- - s_{hj}, \text{ if } s^- > s_{hj} \text{ or} \\ s_{hj} - s^+, \text{ if } s_{hj} > s^+ \text{ or} \\ 0, \text{ otherwise.} \end{cases}$$

For a longer sequence, involving a set of distance values, the cost function is defined as the sum of the individual ones:

$$c_g(\{s_1, s_2, \ldots, s_n\}) = \sum_{i=1}^{n} c_g(s_i).$$

For example, assume the following most specific model for the event Z:

$$\begin{aligned} LGG_s(Z) &= LGG_s(A,B) \wedge LGG_s(B,C) \wedge LGG_s(C,D) \\ &= [0..0] \wedge [0..0] \wedge [0..0]. \end{aligned}$$

Using the six alternative positive sequences in section 4.2 as an example, the cost function is calculated as follows:

$$\begin{aligned} c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (0-(-2)) + (2-0) + (2-0) = 6, \\ c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (0-(-3)) + (2-0) + (2-0) = 7, \\ c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (1-0) + 0 + (2-0) = 3, \\ c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (0-(-2)) + (4-0) + (1-0) = 7, \\ c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (0-(-3)) + (4-0) + (1-0) = 8, \\ c_g(\{s_{AB}, s_{BC}, s_{CD}\}) &= (1-0) + (2-0) + (1-0) = 4. \end{aligned}$$

The best choice is the one that minimises the change to the model, i.e., the third sequence with $c_g = 3$.

The ORAgen algorithm performs a standard *branch-and-bound* search, traversing the *EST* in a top-down *best-first* manner to efficiently arrive at the optimal solution, i.e., minimise c_g. It starts at the root of the tree and generates all of its children, calculating the generalisation cost for each of the generated subsequences. Each subsequence constitutes a *partial solution* to the search task. A list of solutions is maintained, in ascending order of generalisation cost. The least expensive solution, i.e., the head of the list, is removed from the list and expanded further in the same way. The generated solutions are added to the list, maintaining the cost order. The algorithm stops when the head of the solution list contains a complete solution. Algorithm 4.2 provides an overview of the generalisation algorithm.

Input: The LGG_s of the event definition; the terminal subevent occurrence, $e(i^-, i^+)$, representing the positive example.

Output: The new LGG_s or $\{\}$ if it fails to cover the positive example.

ORAgen$(LGG_s, e(i^-, i^+))$ **:**

```
G ← {e(i⁻, i⁺)}          % initialise with the terminal
N1 ← generate-children(e(i⁻, i⁺))
WHILE N1 ≠ {}:
        G ← tail(G)
        G ← append-sorted(G, N1)
        N1 ← generate-children(head(G))
ENDWHILE
IF G = {}: RETURN {}
P ← construct-path[a](e(i⁻, i⁺),head(G))
LGGs ← update-model(LGGs, P)
RETURN LGGs
```

[a]Trace the path from the leaf back to the root.

Algorithm 4.2: The ORAgen for distance-only generalisation.

Figure 4.8 illustrates how the optimal solution in the above example is found with the ORAgen algorithm. The shaded nodes of the tree are the ones that are visited and the highlighted path is the one that is finally selected. The generalisation cost at each individual node[2] is shown in the box above the node and the cost of each path through the tree is shown beneath each of the leaves. The arrows illustrate the sequence of least-cost nodes, as selected by the algorithm.

Finally, the following are a few interesting details of the algorithm:

- The EST does not need to be generated in full. Instead of traversing an already generated EST, the ORAgen algorithm can generate the branches which are useful to the search.
- Each partial solution is represented in the solution list by the first, according to the precedence sequence, subevent occurrence rather than the complete subsequence. If chosen to be expanded, this occurrence is replaced by its children in the tree. Using this representation of solutions, duplicate nodes are not permitted in the list and if more than one partial solutions share the same starting occurrence, the cheapest one is chosen. In this way the traversal of identical branches of the tree is avoided. The

[2]For simplification, the cost associated with a distance between a node and one of its children is assigned to the child. This convention is used throughout the chapter.

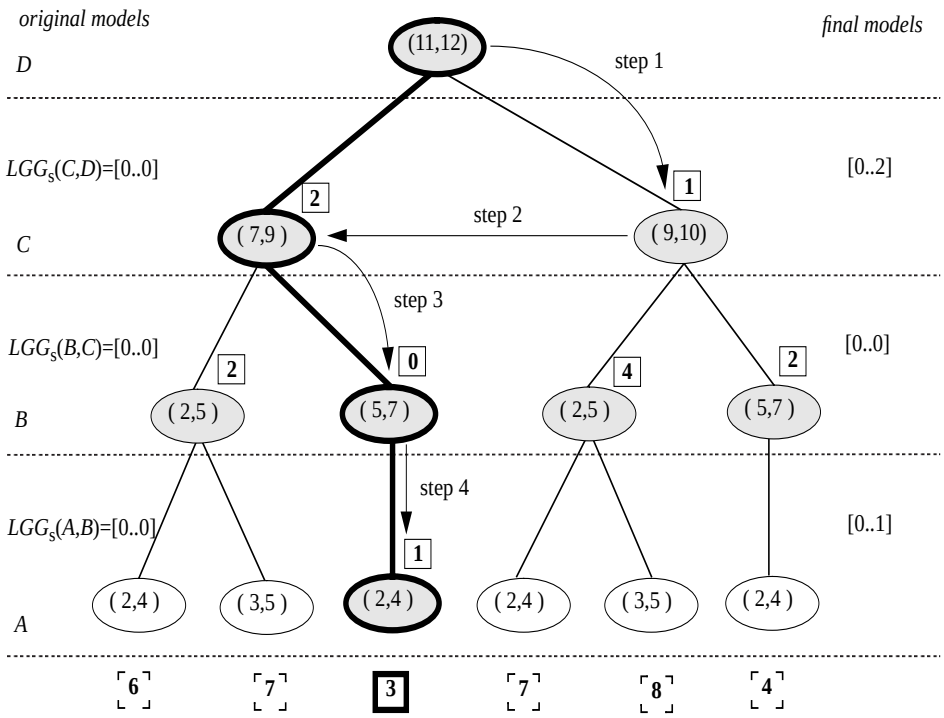


Figure 4.8: Best-first traversal of the example EST. Shaded nodes are the visited ones. Costs of change are shown in the boxes. The selected solution is highlighted.

drawback of this representation is that the path from the selected leaf to the root needs to be reconstructed.

- A subsequence is removed from the list if it cannot lead to a complete sequence, i.e., a node of the tree is pruned when it does not have any children or all its children are pruned.
- Out of the children of a node in the tree, those that satisfy the PPS range are traversed first and the rest are examined only if all others are pruned. This is an additional bias for the original parameter settings, which is not presented in Alg. 4.2.
- Out of two solutions with the same minimum cost, the longest one is selected, because it is closer to a complete solution.

4.4.2 Distance-only specialisation

The specialisation algorithm aims to contract appropriately the MGG_s ranges, in order to exclude negative examples. A negative example is represented by a negative EST, which corresponds to a set of sequences of subevent occurrences. Each such sequence could lead to the recognition of an event occurrence that is not desired. Examples of these are presented in section 4.2, where six sequences of the subevents (A, B, C, D) are presented, which lead to the erroneous recognition of Z occurrences. The problem of optimal specialisation is harder than that of generalisation. This section explains why this is the case and presents an efficient solution to the problem.

The task of specialisation differs in many ways from that of generalisation:

- In specialisation, each of the negative sequences, i.e., each branch of the EST from root to leaf, needs to be excluded. There are no alternative negative examples and therefore no search of the nature performed in ORAgen is needed.
- However, the modification of a single parameter range can exclude one whole branch, because all of the constraints need to be satisfied for the event to be recognised. In other words, a branch can be pruned at any point along its length. Therefore the search required here does not aim to select the most appropriate sequence, but the most appropriate pruning point for each sequence. An optimal solution is sought again and the optimality criterion is of similar flavour to the generalisation cost.
- A further complication comes from the fact that by deciding on a pruning point for one branch in the EST, other branches are affected. The simplest example is the case of branches which have common components, e.g. by pruning the root of the EST all branches are pruned. More complex dependencies between tree branches are examined later in this section.
- Finally, negative ESTs can be significantly larger than positive ones, because the start point of each negative sequence cannot be determined. Therefore, ways of restricting the size of the tree and generating as small a part of it as possible are needed.

In order to simplify the exposition, the assumption is made again that event Z is defined in terms of just two subevents, (A, B). Assume also that $MGG_s(A, B) = [-3..10]$, i.e., distances outside this range are definitely negative, and $LGG_s(A, B) = [0..1]$, i.e., distances inside this range are definitely positive. The following sequence of subevent occurrences is

recognised, which should not lead to the recognition of any Z occurrences:

$$A(2,3), A(4,5), A(6,9), B(7,10).$$

The distance of each of the three A occurrences from $B(7,10)$ is: $(7-3) = 4$, $(7-5) = 2$ and $(7-9) = -2$, respectively. None of these distance values satisfies the $LGG_s(A,B)$ constraint, but all of them satisfy $MGG_s(A,B)$. Therefore, $MGG_s(A,B)$ needs to be contracted to exclude them. The resulting range is $[-1..1]$, which excludes the values $-2, 2$ and 4. Figure 4.9 illustrates the specialisation process considering each of the three negative sequences in turn.

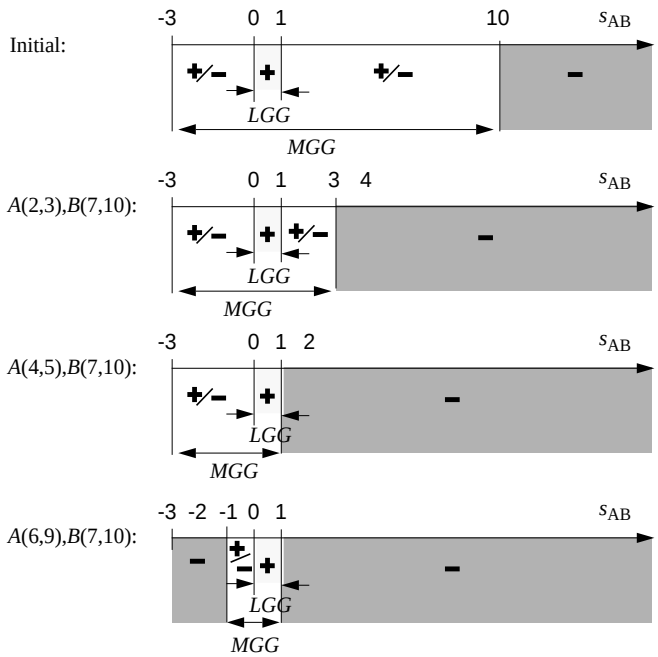


Figure 4.9: One-dimensional incremental specialisation. +: positive range; -: negative range; +/-: region of uncertainty.

The first observation to be made in the above example is that the side on which the contraction takes place depends on the LGG_s model, rather than the MGG_s one. *Left specialisation* takes place, when the distance value is lower than the lower bound of the LGG_s, e.g. $s_{AB} = -2 < 0$ above, and *right specialisation* when the distance value is higher than the high LGG_s bound, e.g. $s_{AB} = 4 > 1$ above. The remaining possibility, which is not illustrated in the above example, is that the distance value satisfies the LGG_s range and can therefore not be

excluded. In that case, the distance value is called *non-excludable*. Non-excludable parameter values limit the pruning choices along a branch of the EST. Thus, a set of values, corresponding to the same distance constraint, can be divided into three subsets:

- The *left specialisation set*, which contains values lower than the low LGG_s bound and requires an increase of the low MGG_s bound, in order to be excluded. The notation L is used in the rest of this section for this set.
- The *right specialisation set*, which contains values higher than the high LGG_s bound and requires a decrease of the high MGG_s bound, in order to be excluded. This set is denoted by R.
- The *non-excludable set* of values, which satisfy the LGG_s range and cannot be excluded. This set is denoted by Q.

Another interesting feature of the problem is that the exclusion of one distance value may cause the automatic exclusion of the other. In the above example, if the distance value $s_{AB} = 2$ is examined first, it causes a decrease of the upper $MGG_s(A, B)$ bound to 1. The distance value $s_{AB} = 4$ will then have no further effect on the $MGG_s(A, B)$ range, since it will be already excluded by it. More generally, it is true that in each of the L and R sets of distance values, there is one which causes the largest contraction of the MGG_s and the automatic exclusion of all others. This value is the one closest to the LGG_s range. The idea of automatic exclusion plays a central role in the design of the ORAspec specialisation algorithm, because it imposes an ordering on the values of the L and R sets. This ordering is termed *exclusion* and is denoted by $>_e$. Given two distance values s_h, s_j from one of the L and R sets, $s_h >_e s_j$ if the exclusion of s_h causes the automatic exclusion of s_j. The exclusion order determines the effect of a pruning decision by the ORAspec algorithm.

The concept of exclusion can be illustrated better with the use of a longer event definition, such as the original one used for Z, i.e., in terms of the subevents (A, B, C, D). For the sake of simplicity, assume that the most specific model is:

$$LGG_s(A, B) = LGG_s(B, C) = LGG_s(C, D) = [0..0]$$

and the most general one is:

$$MGG_s(A, B) = MGG_s(B, C) = MGG_s(C, D) = [-10..10].$$

Assume also the following sequence of subevent occurrences:

$$A(1,2), A(3,5), B(4,6), B(5,7), C(7,8), C(12,14), D(12,15),$$

which should not cause the recognition of any Z occurrence. The negative *EST* for this example is shown in Fig. 4.10. There are three types of edges in the tree, corresponding to the three different sets, in which distance values are split. For example, the two children of the root node correspond each to a different set, because the corresponding s_{CD} values lie on either side of the $LGG_s(C,D)$ range. $C(12,14)$ overlaps $D(12,15)$, with a distance value of $s_{CD} = 12 - 14 = -2$, while the distance of $C(7,8)$ from $D(12,15)$ is $s_{CD} = 12 - 8 = 4$. Similarly, the distance between $B(5,7)$ and $C(7,8)$, $s_{BC} = 7 - 7 = 0$, cannot be excluded, because it falls in the $LGG_s(B,C)$ range.

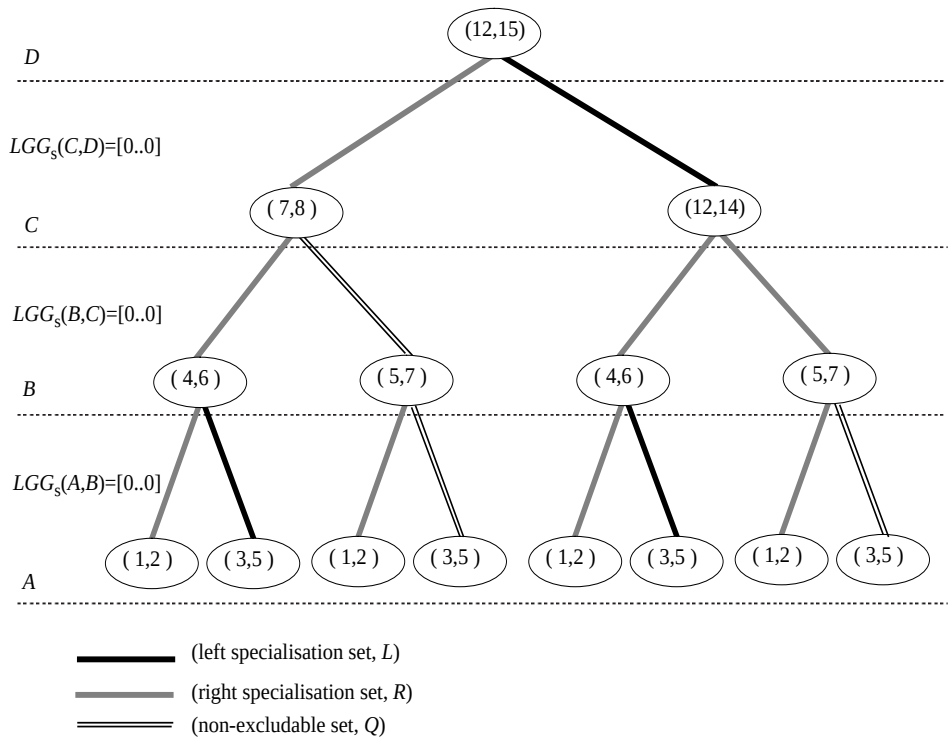


Figure 4.10: A negative EST including information about the three specialisation sets.

One of the eight sequences of the EST in Fig. 4.10 can only be pruned at the level of the s_{CD} parameter. This is the sequence:

$$A(3,5), B(5,7), C(7,8), D(12,15).$$

Thus, the upper bound of the $MGG_s(C, D)$ range needs to be reduced to exclude the distance value $s_{CD} = 12 - 8 = 4$. This modification causes the automatic exclusion of four sequences, i.e., the ones on the left branch below the root. The pruning of the right branch is more complex. All sequences can be pruned in more than one way and automatic exclusion has to be taken into account. For example, the two occurrences of B, $B(4, 6)$ and $B(5, 7)$, belong to the same specialisation set, R, since their distance values, $s_{BC} = 12-6 = 6$ and $s_{BC} = 12-7 = 5$, both require the reduction of the upper $MGG_s(B, C)$ bound. Note that according to the exclusion ordering of R, $s_{BC} = 5 >_e s_{BC} = 6$, i.e., the smaller distance value, requires a larger change of the $MGG_s(B, C)$ bound and causes the automatic exclusion of the other. Automatic exclusion is not restricted to sibling nodes. It applies to all distance values at the same level of the EST, since they are all affected by a change to the same MGG_s range. For example, the distance values between $A(1, 2), B(4, 6)$ and $A(1, 2), B(5, 7)$ are $s_{AB} = 4 - 2 = 2$ and $s_{AB} = 5 - 2 = 3$ respectively. They are both in the R specialisation set and any choice about one might affect the other, although they do not correspond to sibling nodes. Thus, if $s_{AB} = 2$ is excluded, $s_{AB} = 3$ will also be excluded automatically.

The goal of specialisation is to prune each EST branch, i.e., all the leaves of the tree. There are two types of pruning involved: *explicit* and *implicit*. A leaf node is pruned explicitly, when a specialisation decision is made aiming to exclude one of the distance values on the branch from the root to this leaf. An example of this was seen above, where the decision was made to reduce the upper bound of $MGG_s(C, D)$ to exclude $s_{CD} = 4$. This distance value is on the branch of four leaves, which are pruned explicitly. Implicit pruning takes place, with the use of the exclusion order at each level of the tree. When a decision is made to exclude one distance value, a number of other values at the same level will be excluded, as was seen above for the $s_{BC} = 5$ and $s_{BC} = 6$ distances. The leaves corresponding to the branches of the automatically excluded distances are said to be pruned implicitly. Thus, if $MGG_s(B, C)$ is modified to exclude $s_{BC} = 5$, $s_{BC} = 6$ is also excluded and the two leaf nodes below it are pruned implicitly.

Because there is clearly more than one solution to the decision of pruning a negative EST, a search needs to be performed to select the best solution. The optimality criterion for this search is a *specialisation cost* function, which is similar to the generalisation cost function, in the sense that it provides a bias for minimum model change. Given a distance value s_{hj}, between

two subevent occurrences and the ranges $LGG_s(e_h, e_j) = [s_l^- .. s_l^+]$ and $MGG_s(e_h, e_j) = [s_m^- .. s_m^+]$, the generalisation cost function associated with the explicit pruning of this value is defined as:

$$c_s(s_{hj}) = \begin{cases} s_m^+ - (s_{hj} - 1), \text{ if } s_{hj} > s_l^+ \text{ or} \\ (s_{hj} + 1) - s_m^-, \text{ if } s_l^- > s_{hj}. \end{cases}$$

The c_s function is not defined inside the LGG_s range, because the distance value is non-excludable, and does not need to be considered outside the MGG_s, where no specialisation is needed. For a set of pruning choices the cost function is defined as the sum of the individual ones:

$$c_s(\{s_1, s_2, \ldots, s_n\}) = \sum_{i=1}^{n} c_s(s_i).$$

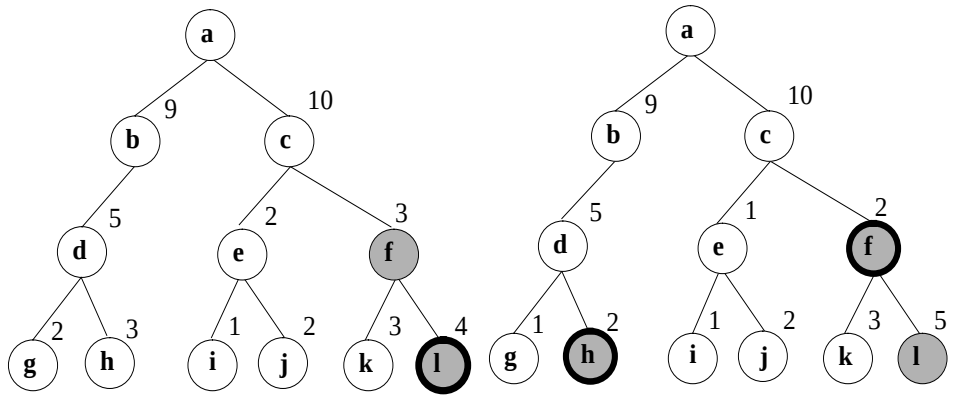
As an example, the EST of Fig. 4.10 can be pruned by explicit pruning of $s_{CD} = 12 - 8 = 4$, which prunes the left half of the tree and $s_{BC} = 12 - 7 = 5$, which prunes the right part. The cost of this solution would be:

$$\begin{aligned} c_s(\{s_{CD} = 4, s_{BC} = 5\}) &= c_s(s_{CD} = 4) + c_s(s_{BC} = 5) = \\ &\quad (10 - (4 - 1)) + (10 - (5 - 1)) = 13, \end{aligned}$$

where 10 is the upper bound of $MGG_s(C, D)$ and $MGG_s(B, C)$.

In order to illustrate the complexity of finding the optimal specialisation solution, consider the EST of Fig. 4.11(a). The time stamps in the nodes have been replaced by arbitrary letters to simplify the explanation. For the same reason, the assumption is made that all distance values belong to the same specialisation set, i.e., they lie on the same side of the LGG_s ranges. Thus, all distance values at the same level of the EST are related according to the exclusion ordering. The specialisation cost for explicitly excluding each value is attached to the corresponding node. If a decision is made to prune explicitly the node with the highest cost, all other nodes at the same level of the tree are pruned implicitly. In other words, the first three obvious solutions to the problem consist of pruning the most expensive node in each of the three levels. The costs associated with these three solutions are: 10, 5 and 4. The third of these choices, i.e., explicitly pruning the node **l** is also the optimal solution in this case. If the branch of the **l** node, (**a,c,f,l**), is considered in isolation, this is a suboptimal choice, since it is less expensive to prune the branch at **f**. If this choice was made, **k,l** would be pruned explicitly and **i,j** implicitly. However,

the nodes **g,h** would remain unpruned and additional cost would be incurred in order to prune them.



(a) Pruning at a single level. (b) Pruning at multiple levels.

Figure 4.11: Two complex specialisation examples. All nodes belong to the same specialisation set. Their costs are shown above them. Shaded nodes are considered as pruning choices in the text. The highlighted nodes provide the optimal solution.

Minimisation of the maximum cost at each level of the EST is not always the best solution, though. In Fig. 4.11(b), some of the pruning costs have been changed and pruning node **l** is no longer the optimal choice. Instead, it is preferable to prune the branch of **l** at **f**, which causes explicit pruning of **k,l** and implicit pruning of **i,j**. The cost of pruning the remaining two sequences, **g,h**, is only 2, i.e., the cost of explicitly pruning **h**, which also causes the implicit pruning of **g**. The overall cost of this solution is $c_s(\{\mathbf{f,h}\}) = 2 + 2 = 4$, which is less than that of pruning **l**.

The number of alternative solutions for pruning an EST increases exponentially with its size. In fact, for a tree of height h, i.e., an event-definition containing h subevents and a branching factor of b, i.e., each node having b children, the number of solutions is given by the

$(h-1)$th term of the following sequence (see appendix B):[3]

$$\begin{aligned} S_1 &= 1 \\ S_2 &= b+1 \\ S_n &= \frac{1-S_{n-1}^{(b+1)}}{1-S_{n-1}}. \end{aligned}$$

In order to gain an idea of the combinatorial explosion to which this result corresponds, Tab. 4.1 gives some indicative sizes of the search space for different values of h and b. Even for relatively small trees the search space can be enormous. For this reason, it is desirable to devise an algorithm which finds the optimal choice, without enumerating all possible solutions.

height(h)	Branching factor (b)		
	2	3	4
2	1	1	1
3	3	4	5
4	13	85	781
5	183	$\sim 621.5 \times 10^3$	$\sim 372.5 \times 10^9$

Table 4.1: Indicative search space sizes for the specialisation of distances.

The ORAspec algorithm, differs from the branch-and-bound ORAgen algorithm in that the first complete solution is already known at the root of the tree, i.e., to prune explicitly the most expensive child of the root, node **c** in Fig. 4.11(a). The algorithm starts from that point and tries to construct alternative solutions, following the most expensive path to the leaf nodes. This may seem unintuitive, but it is explained by the fact that the most extensive modification of the model guarantees the pruning of *all* of the less expensive nodes at the same level. An extreme example is shown in Fig. 4.11(a), where the optimal solution is the one with the highest cost at that level. Thus, in the EST of Fig. 4.11(a) ORAspec traverses the path (**a,c,f,l**) first. Each time the algorithm moves from one level of the tree to the lower one, an alternative way of pruning the siblings at the previous level must be found. In the above example, this leads to the following three solutions:

[3]This is a worst-case estimate.

Solution	Pruning choices	**low-cost**	**high-cost**
1	**c**,[b]	10	10
2	**f**,(**b**),[e]	3	$3 + 9 = 12$
3	**l**,(**e,b**),[k]	4	$4 + 2 + 9 = 15$

Nodes in round brackets are non-leaves, which have not been expanded, i.e., their children have not been examined. Nodes in square brackets are the ones which have been examined, but found to be pruned implicitly by other nodes in the solutions. Note that the effect of implicit pruning on nodes which have not been examined is not known. For instance, it is not known that by pruning **l** all leaf nodes below **b** and **e** are also pruned and that **b,e** do not need to be pruned explicitly. As a result, the actual cost of the generated solutions is not known. However, a lower and an upper bound to the cost can be calculated. The lower bound makes the optimistic assumption that the selected choice prunes the whole tree. The upper bound makes the assumption that it only prunes its siblings and that the alternative nodes in each visited level – the nodes in round brackets above – need to be pruned explicitly. Once the leaf node has been reached, the solution with the lowest **low-cost** bound is examined. If its **high-cost** equals its **low-cost**, then this is the optimal solution. Otherwise, ORAspec attempts to decrease the upper bound by finding alternative ways to achieve the effect of the most expensive pruning choice. In the example, solution 2 has the lowest **low-cost** and its most expensive pruning choice is **b**. Thus, **b** is selected to be examined further, in the same way as above, i.e., following the most expensive path to the leaves. The updated list of solutions is:

Solution	Pruning choices	**low-cost**	**high-cost**
1	**c**,[b]	10	10
2	**f,b**,[e]	$3 + 9 = 12$	$3 + 9 = 12$
3	**l,b**,(**e**),[k]	$4 + 9 = 13$	$4 + 2 + 9 = 15$
4	**d**,[e,f]	5	5
5	**f,h**,[e,g]	$3 + 3 = 6$	$3 + 3 = 6$
6	**l,d**,[e,k,f]	$4 + 5 = 9$	$4 + 5 = 9$
7	**l**,(**e**),[g,h,k]	4	$4 + 2 = 6$

Note that as the nodes below **b** are visited, all solutions which include **b** are updated. These are solutions 2 and 3. The same process needs to be repeated again, because solution 7, which has the lowest **low-cost**, does not have an equally low **high-cost**. Thus, **e** needs to be explored,

leading to the final list of solutions:

Solution	Pruning choices	**low-cost**	**high-cost**
1	**c**,[b]	10	10
2	**f,b**,[e]	$3+9=12$	$3+9=12$
3	**l,b,e**,[k]	$4+2+9=15$	$4+2+9=15$
4	**d**,[e,f]	5	5
5	**f,h**,[e,g]	$3+3=6$	$3+3=6$
6	**l,d**,[e,k,f]	$4+5=9$	$4+5=9$
7	**l,e**,[g,h,k]	$4+2=6$	$4+2=6$
8	**l,b**,[i,j,k]	$4+9=13$	$4+9=13$
9	**l**,[g,h,i,j,k]	4	4

Solution 9 is clearly the optimal one.

Due to the complexity of the examined example, the algorithm needs to traverse the whole EST, before the optimal solution is decided. A more realistic, but still complex, example is the one of Fig. 4.10. The traversal of this tree is shown in Fig. 4.12. The algorithm first follows the rightmost path to leaf $A(3,5)$ and then moves to $C(7,8)$ and examines the most expensive branch below it. This is the branch containing the two non-excludable distances. The optimal choice is the one explicitly pruning $B(5,7)$ and $C(7,8)$, i.e., distances $s_{BC} = 12 - 7 = 5$ and $s_{CD} = 12 - 8 = 4$. Each of these causes a reduction of the upper limit of the corresponding MGG_s range, leading to the updated ranges shown on the right of the EST.

The use of the three separate specialisation sets, L, R, Q, adds a further complication to the specialisation algorithm. Each time a node is replaced by its children, three separate sets are generated. The nodes in the Q set are examined first, because they need to be pruned further down the tree. If they cannot be pruned, the other two sets of sibling nodes do not need to be considered. Nodes in L and R need to be pruned separately and therefore two separate node lists need to be maintained for each solution. Algorithm 4.3 provides an overview of the ORAspec algorithm. A proof of the optimality of the algorithm is presented in appendix C.

4.5 Extending to refinement of duration constraints

In the discussion so far it has been assumed that duration constraints are fixed. If this assumption is relaxed, the dimensionality of the refinement task increases. In addition to modifying

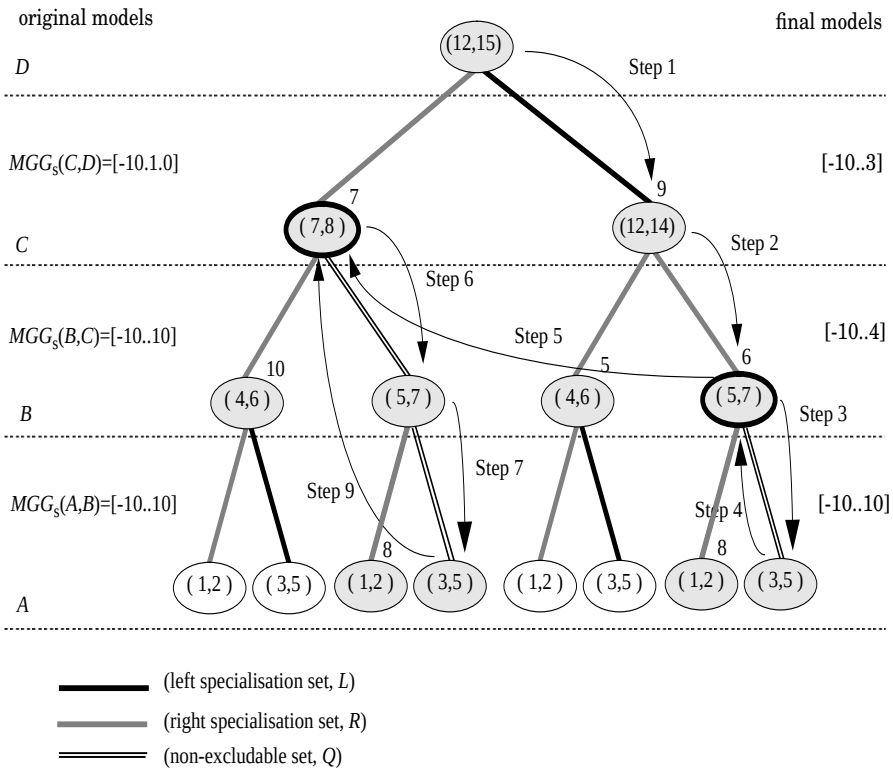


Figure 4.12: Trace of the specialisation algorithm. The arrows show the traversal of the algorithm. The two highlighted nodes are the ones selected for pruning.

the most general and most specific models for the distance parameters, i.e., LGG_s and MGG_s, the models for the duration parameters, LGG_d and MGG_d need to be modified when a new positive or negative example is received. This extension of the problem has little impact on the ORAgen generalisation algorithm, examined in section 4.5.1, but it makes the specialisation search significantly more complex, as seen in section 4.5.2.

4.5.1 Extended generalisation algorithm

Given an EST representing a set of alternative positive examples, the objective of the generalisation search is to select the sequence which causes the least change to the original model. In section 4.4.1, it was shown how a simple branch-and-bound search, using the generalisation cost heuristic, can find the optimal solution to this problem. The only extension that is needed, in order to take the duration constraints into account, is in the definition of the cost function.

Input: The MGG_s of the event definition; the terminal subevent occurrence, $e(i^-, i^+)$.
Output: The new MGG_s or $\{\}$ if it fails to exclude the negative example.
Globals: The LGG_s model.
ORAspec$(MGG_s, e(i^-, i^+))$ **:**

```
G ← {}                                          % initialise solution list
n ← e(i⁻, i⁺)                                   % current node examined
WHILE n ≠ {}:                                   % while optimal solution not found
        (L, Q, R) ← generate-children(n, LGG_s)
        G ← update-solutions^a(L, Q, R, G)      % update the solution list
        IF L = Q = R = {}:                      % leaf node
                s ← min-cost(G)                 % least-cost solution so far
                IF (low-cost(s) = high-cost(s)): n ← {}      % optimal solution
                ELSE: n ← max-cost^b(s)         % most expensive pruning choice
                                                % in the solution
        ELSE:                                   % non-leaf node
                s ← build-solution(L, R)
                n ← max-cost(s)                 % most expensive path to leaves
        ENDIF
ENDWHILE
IF G = {}:MGG_s ← {}
ELSE: MGG_s ← update-model(s, MGG_s)            % incorporate solution to MGG_s
RETURN MGG_s
```

[a]The updating of the solution list involves the replacement of each solution, which contains the expanded node, by a pair of solutions: one in which this node is pruned explicitly and one in which it is replaced by its children.

[b]If there are non-excludable nodes they have the highest cost.

Algorithm 4.3: An overview of the ORAspec algorithm for distance-only specialisation.

Each branch of the EST, corresponds to a sequence of subevent occurrences, some of which may not satisfy the LGG_d model. The generalisation cost for changes to the LGG_d is defined in the same way as for changes to LGG_s. If $LGG_d = [d^-..d^+]$ and d_h the duration of a subevent occurrence,

$$c_g(d_h) = \begin{cases} d^- - d_h, \text{ if } d^- > d_h \text{ or} \\ d_h - d^+, \text{ if } d_h > d^+ \text{ or} \\ 0, \text{ otherwise.} \end{cases}$$

The overall generalisation cost for a sequence is extended to include the cost of generalising the LGG_d ranges:

$$c_g(\{s_1, s_2, \ldots, s_n\}, \{d_1, d_2, \ldots, d_m\}) = \sum_{i=1}^{n} c_g(s_i) + \sum_{j=1}^{m} c_g(d_j).$$

The assumption that is made here is that the marginal change in the duration ranges is of the same scale as the change in the distance ranges. Using this cost function, the ORAgen algorithm, in Alg. 4.2 can be used for the generalisation of both distance and duration ranges.

For example, assume again that event Z is defined as the sequence (A, B, C, D) and the most specific model for the temporal parameters of the definition is as follows:

$$\begin{array}{rcccccccc} LGG_s(Z) & = & LGG_s(A,B) & \wedge & LGG_s(B,C) & \wedge & LGG_s(C,D) & & \\ & = & [0..0] & \wedge & [0..0] & \wedge & [0..0] & & \\ LGG_d(Z) & = & LGG_d(A) & \wedge & LGG_d(B) & \wedge & LGG_d(C) & \wedge & LGG_d(D) \\ & = & [1..1] & \wedge & [1..1] & \wedge & [1..1] & \wedge & [1..1]. \end{array}$$

Assume also the following sequence of subevent occurrences:

$$A(2,3), A(3,5), B(4,5), B(5,6), C(6,9), C(8,10), D(10,12),$$

which should cause the recognition of $Z(?, 12)$.[4] The EST for this example and its traversal by the generalisation algorithm is shown in Fig. 4.13. The generalisation cost associated with each node consists now of two components: the cost of generalising its distance and the cost of generalising its duration constraint. Shaded nodes are the ones generated and the highlighted ones correspond to the sequences selected by ORAgen. In this case there are two sequences with identical cost and an arbitrary choice between them needs to be made. Note that the generalisation cost of the root is unavoidable and does not affect the search.

4.5.2 Extended specialisation algorithm

Adding the duration parameters in the specialisation search increases significantly the dimensionality of the search space. The reason for this is that there are now several more pruning options along each branch of the EST, since each node can be pruned either by duration or by distance. As a simple example, assume the two-subevent definition of Z in terms of (A, B) and the following sequence of subevent occurrences, which should not lead to the recognition of Z: $A(2,3), B(8,12)$. Assume also that the most general and most specific models for Z are as follows:

$$MGG_d(A) = MGG_d(B) = [1..10], \qquad MGG_s(A,B) = [-10..10]$$
$$LGG_d(A) = LGG_d(B) = [5..5], \qquad LGG_s(A,B) = [-1..1]$$

[4]The symbol '?' denotes that the begin point of the event is not defined in the training feedback.

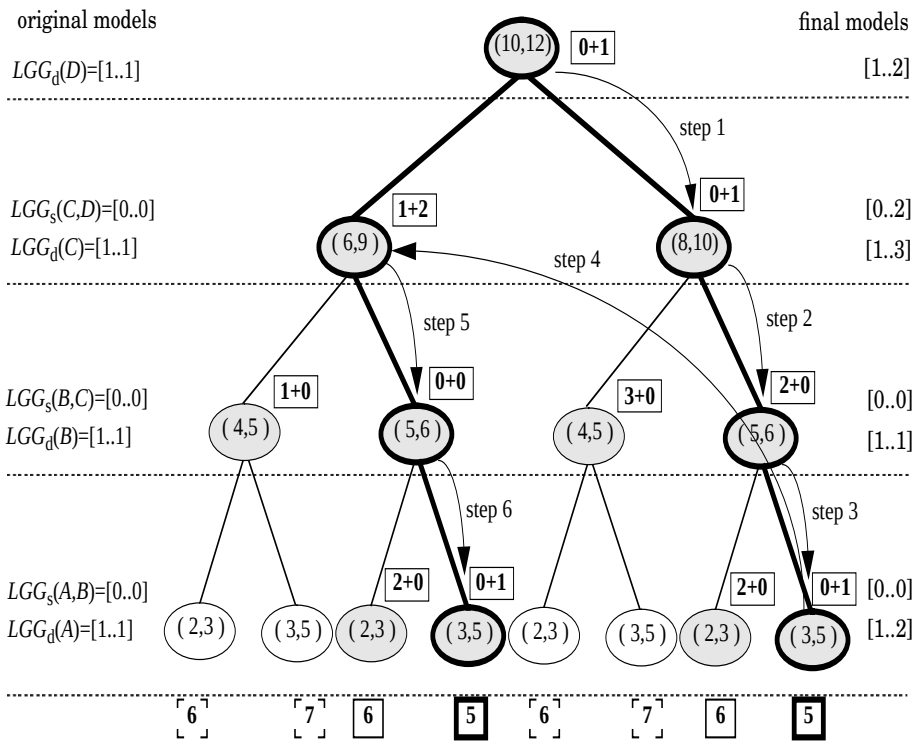


Figure 4.13: Trace of ORAgen refining both duration and distance.

The distance and duration values for the subevent sequence is:

$$d_A = 3 - 2 = 1, s_{AB} = 8 - 3 = 5, d_B = 12 - 8 = 4,$$

all of which satisfy the MGG but not the LGG models. In order to exclude this negative sequence, any of the MGG ranges may be modified. The most reasonable choice, i.e., the one causing the minimum change to the MGG models, is to increase the lower bound of $MGG_d(A)$ from 1 to 2, so that it is no longer satisfied by $d_A = 1$.

However, when the EST contains more than a single branch, the optimal pruning choice for all branches becomes more complex. In the above example, if the following sequence of subevent occurrences is recognised instead:

$$A(2,3), A(6,10), B(8,12),$$

there will be two negative sequences, which need to be excluded:

$$A(2,3), B(8,12) \text{ and } A(6,10), B(8,12).$$

with the corresponding parameter values:

$$d_A = 3 - 2 = 1, \quad s_{AB} = 8 - 3 = 5, \quad d_B = 12 - 8 = 4,$$
$$d_A = 10 - 6 = 4, \quad s_{AB} = 8 - 10 = -2, \quad d_B = 12 - 8 = 4.$$

The best pruning choice is less clear in this case. The options are to:

1. increase the low bound of $MGG_d(B)$ to 5 to exclude $d_B = 4$,
2. increase the low bound of $MGG_d(A)$ to 5 to exclude both $d_A = 1$ and $d_A = 4$.
3. increase the low bound of $MGG_s(A, B)$ to -1 to exclude $s_{AB} = -2$ and decrease the upper bound of $MGG_s(A, B)$ to 4 to exclude $s_{AB} = 5$.
4. increase the low bound of $MGG_d(A)$ to 2 to exclude $d_A = 1$ and the low bound of $MGG_s(A, B)$ to -1 to exclude $s_{AB} = -2$.

In this case, the first two options cause the minimum overall change to the models. The additional complexity of the problem stems from the numerous pruning combinations for the nodes in a single level of the tree, occurrences of subevent A in this example. In the distance-only specialisation, examined in section 4.4, the most expensive pruning choice was necessary and sufficient for that purpose. The fourth option above illustrates that this is no longer the case. Increasing the low bound of $MGG_d(A)$ to 2 is not the most expensive specialisation option and only prunes one of the two occurrences of A. The other is pruned by the change to $MGG_s(A)$. The search for the optimal pruning solution within a single level of EST nodes is the only extension of the ORAspec algorithm, needed for the handling of duration. This is a complex problem and it is the main focus of this section. The basic ORAspec algorithm, as described in Alg. 4.3, does not need to be changed.

The increased dimensionality of the specialisation problem has a significant effect on the size of the search space. The search space for a tree of height h and branching factor b is now given by the hth term of the following sequence:[5]

$$\begin{aligned} S_1 &= 1 \\ S_2 &= b + 1 \\ S_n &= \sum_{i=1}^{n} i S_{n-1}(1 + S_{n-1})^{n-i}. \end{aligned}$$

To illustrate the combinatorial explosion corresponding to this result, Tab. 4.2 presents the size

[5] Actually it is $S_h + 1$ (see appendix B).

of the space for small values of b and h. The combinatorial explosion is clearly much higher for this problem than for the distance-only specialisation (see Tab. 4.2).

height(h)	Branching factor (b)		
	2	3	4
2	3	4	5
3	22	157	1556
4	508	389.5×10^4	586.5×10^{10}

Table 4.2: Indicative search space sizes for the specialisation of distance and duration.

The large exponential increase to the size of the search space is due to the additional pruning choices which have to be considered. As mentioned above, this increase is caused by the numerous alternative pruning combinations for nodes in the same level of the EST. In the simplest case, these nodes are siblings and for the sake of simplicity they will be referred to as siblings in the rest of this discussion. Since the most expensive pruning combination is no longer the only option for sibling nodes, a search has to take place for an alternative combination which achieves the same effect, but minimises the specialisation cost. The space for this search can be quite large and for this reason an efficient search algorithm is needed.

In the distance-only specialisation, the notion of left and right specialisation was introduced, meaning that specialisation affected the lower or the upper bound of the most general model, respectively. A set of sibling nodes was then divided into three mutually exclusive subsets, (L, Q, R), using the most specific model. The nodes which satisfied the LGG model were named non-excludable and grouped in set Q. A similar separation of sibling nodes is possible here too, but there are five mutually exclusive sets involved:

- Left specialisation by distance (L_s).
- Right specialisation by distance (R_s).
- Left specialisation by duration (L_d).
- Right specialisation by duration (R_d).
- Non-excludable (Q).

The allocation of nodes into the duration and distance sets is done on the basis of the most preferable pruning choice for a node, i.e., the least expensive one. The separation between left and right sets is done with the use of the LGG model, as before. Non-excludable are now the

nodes which cannot be pruned by either of the two parameters. In the simple example shown above, there are two sibling A nodes, i.e., $A(2,3)$ and $A(6,10)$. The duration and distance values corresponding to them are $(d_A = 1, s_{AB} = 5)$ for the first and $(d_A = 4, s_{AB} = -2)$ for the second. The associated specialisation costs are $c_s(d_A) = (1+1) - 1 = 1, c_s(s_{AB}) = 10-(5-1) = 6$ and $c_s(d_A) = (4+1)-1 = 4, c_s(s_{AB}) = (-2+1)-(-10) = 9$. Therefore the preferred choice for $A(2,3)$ is specialising MGG_d, i.e., duration pruning, and the same for $A(6,10)$. Furthermore they both cause an increase in the low bound of MGG_d and therefore belong to L_d. The other four specialisation sets are empty for this example.

In order to draw the attention on the essence of the problem, in the rest of this section costs are used instead of the underlying temporal parameters. Thus, a sibling will be represented by the following notation:

$$(N, C_s, P_s, C_d, P_d),$$

where N is a node, like $A(2,3)$, C_s the cost of pruning it by distance, P_s an indicator of whether it is pruned left or right by distance and C_d, P_d the cost and pruning side for duration. Thus, the above two examples would be represented by:

$$(A(2,3), 6, r, 1, l) \text{ and } (A(6,10), 9, l, 4, l),$$

meaning that the first node is pruned right by distance and left by duration and the second node left by distance and by duration. Moreover, small letters will be used to denote nodes, rather than the actual event occurrences, e.g. $(a, 6, r, 1, l)$ and $(b, 9, l, 4, l)$, where a, b are sibling nodes. Using this representation, the problem to be solved is as follows:

> Given a set of siblings S, separated into five mutually exclusive subsets $S = L_s \bigcup R_s \bigcup L_d \bigcup R_d \bigcup Q$, find a subset of these S', which determine the least expensive way of excluding all the nodes in $S \backslash Q$. The pruning choice determined by S' is represented as a quadruple of nodes: $(E_{sl}, E_{sr}, E_{dl}, E_{dr})$, one for each of the four types of specialisation: L_s, R_s, L_d, R_d. The symbol '$-$' denotes that no specialisation is needed on that side of the MGG range.

Thus, in the above example,

$S = \{(a, 6, r, 2, l), (b, 9, l, 4, l)\} =$

$$L_s = \{\} \bigcup R_s = \{\} \bigcup L_d = \{(a, 6, r, 2, l), (b, 9, l, 4, l)\} \bigcup R_d = \{\} \bigcup Q = \{\}$$

and the minimum cost choice is to prune both nodes by duration. Therefore: $S' = \{(b, 9, l, 4, l)\}$ contains the node with the highest duration specialisation cost and the pruning choice is represented by $(-, -, b, -)$, meaning that only left duration specialisation is needed and the chosen node for this is b.

In order to simplify the problem, initially the following assumptions will be made:

- Pruning can be done on only one side. Thus, instead of L_s, R_s, L_d and R_d, only two sets S and D are used. Note that this is different from the simpler case examined in distance-only specialisation, where the two sets involved, (L, R), corresponded both to distance specialisation and were independent in terms of pruning choices. Here, all the members of S can also be pruned by duration and vice versa. The separation is done on the basis of the cheapest option for each individual node.
- There are no illegal choices. Hence, Q is empty and every node in S and D can be pruned both by distance and duration.

Both these assumptions are relaxed later in the section.

To gain a better insight on the properties of the problem, assume a set of siblings, separated into:

$$S_1 = \{(a, 9, 21), (b, 7, 12), (c, 5, 15)\}, D_1 = \{(d, 11, 10), (e, 20, 9), (f, 12, 5)\},$$

where each triple holds the node, the cost of pruning by distance and the cost of pruning by duration. Note that the side of specialisation is omitted from the notation, since it is assumed that all specialisation is done on the same side. In this example, the optimal choice is (a, d), i.e., to prune a and d by their preferred options. This combination costs $c_s((a, d)) = 9 + 10 = 19$, which is less expensive than all other choices. A special property of the two sets, which simplifies the decision, is that none of the nodes in S_1 can be pruned by the most expensive duration in D_1 and vice versa. In other words, the alternative pruning choice for each node is higher than the preferred one for nodes in the other set. For example, pruning d, e, f by distance costs $11, 20, 12$ respectively, while the highest distance cost in S_1 is only 9 for a. The term *independent* will be used for sets which have this property. Independent sibling sets are very useful in the design of the algorithm presented below. Another reason why the least-cost pruning combination can easily be found in this example is that all the nodes in S_1 are excluded by a. In other words, both the duration- and distance-pruning cost of b and c are lower than those of a. This is not the case for D_1, where none of d, e, f excludes each other. D_1 is said to

be *minimal*, because all of the nodes in the set can affect the optimal pruning choice, while S_1 is not. Minimality is not essential for the proposed algorithm, but it can improve its performance.

The most expensive nodes of S and D are not always the optimal combination. Consider for example the following sets:

$$S_2 = \{(a, 6, 7), (b, 5, 9), (c, 3, 5)\}, D_2 = \{(d, 7, 5), (e, 7, 4), (f, 20, 3)\}.$$

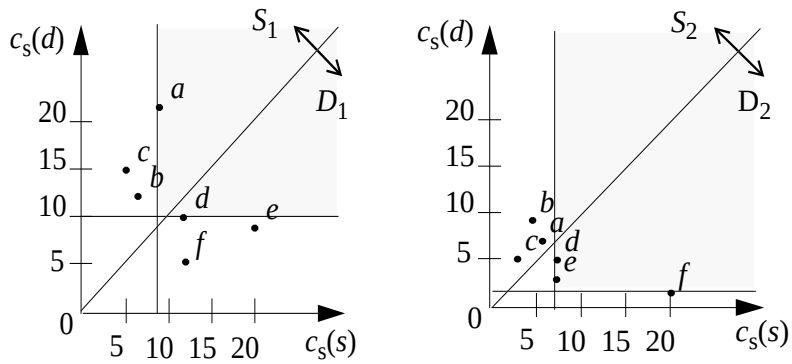
Pruning a and d by the preferred choices in this case leads to a cost $c_s((a, d)) = 6 + 5 = 11$, which is higher than pruning d by distance and f by duration, $c_s((d, f)) = 7 + 3 = 10$. Thus, despite the fact that the duration-pruning cost of d is lower than its distance-pruning cost, overall it is beneficial to prune d by distance, because its distance-pruning cost is not much higher than the distance-pruning cost of the nodes in S_2. Therefore $\{a, b, c, d, e\}$ are pruned by distance and only f needs to be pruned by duration, because its distance-cost is too high. Another interesting property of S_2 and D_2 is that they are not *independent*. Node c is excluded by d, both by duration and by distance, i.e., if d is pruned then c is also certainly pruned. For this reason it plays no role in the decision about the optimal pruning choice. Moreover, neither S_1, nor D_2 are minimal, because a and b exclude c and d excludes e.

Figures 4.14(a) and 4.14(b) present the two sets of nodes in the two-dimensional cost space. The shaded area is not pruned and is the one being maximised by the least expensive pruning combination. Note also that the $45°$ line separates the set of nodes into the S and D sets.

The search algorithm for the minimum cost solution to pruning a set of sibling nodes uses the *independence* of sibling subsets. If a node n_i is excluded by another n_j, both by duration and by distance, it does not affect the decision about the overall optimal pruning choice. The optimal solution should prune n_j, either by duration or by distance and therefore it will automatically prune n_i, too. Thus, excluded nodes can be removed from the sibling subsets and the remaining subsets will be independent and minimal. In the second example above, only c and e are excluded and can be removed from S_2 and D_2 respectively:

$$S_3 = \{(a, 6, 7), (b, 5, 9)\}, D_3 = \{(d, 7, 5), (f, 20, 3)\}.$$

Given two independent sibling subsets, the first two obvious choices is to prune them all either by distance or by duration. These choices for S_3 and D_3 will be $(f, -)$, i.e., pruning by the largest distance cost, which is that of f, and $(-, b)$, i.e., pruning b, which has the largest duration cost. The latter option is also the optimal solution in this case. The first effect of the



(a) Example 1, sets S_1, D_1.

(b) Example 2, sets S_2, D_2.

Figure 4.14: Sibling nodes in the cost space. $c_s(d)$: cost of pruning a node by duration, $c_s(s)$: cost of pruning a node by distance.

independence property is that these two solutions can be found by examining only one of the two subsets. The distance-pruning cost of all the nodes in S_3 is lower than that of the nodes in D_3 and the opposite holds for the duration-pruning cost. In other words, given two independent subsets S and D, the following holds:

$$\forall n_i \in D, n_j \in S : (d(n_i) < d(n_j)) \wedge (s(n_i) > s(n_j)), \tag{4.1}$$

where $d(n)$ and $s(n)$ denote the duration- and distance-pruning costs for n.

The alternative to these two options is to use a two-node pruning combination, such as the combinations: (a, d) and (d, f) examined above. For this type of pruning it can be shown that the optimal solution is either the combination of the two most expensive nodes from each set, pruned by their preferred choice, i.e., the choice (a, d) above, or a combination of nodes from the same set. In other words, there is no need to examine combinations of nodes from the two sets, other than the most expensive combination. The proof for this is presented in appendix C and it makes use of the property described in (4.1) above. Thus, the five possibilities for the optimal solution are the following:

1. $(n_i, -), n_i \in D \wedge \forall n_k \in D : s(n_i) \geq s(n_k)$ or
2. $(-, n_j), n_j \in S \wedge \forall n_l \in S : d(n_j) \geq d(n_l)$ or
3. $(n_i, n_j), n_i \in S \wedge n_j \in D \wedge \forall n_k \in S : s(n_i) \geq s(n_k) \wedge \forall n_l \in D : d(n_j) \geq d(n_l)$ or
4. $(n_i, n_j), n_i, n_j \in S \vee n_i, n_j \in D$.

Case 3 corresponds to the choice (a, d), which prunes all nodes by their preferred choice. If a is pruned by duration instead of distance, then all the nodes in D_3 will also be pruned, since the duration cost of a is higher than theirs. This is guaranteed by the independence property, as presented in (4.1). Thus, the nodes in D_3 can be ignored and a way of pruning the remaining nodes in S_3, e.g. pruning b by distance, needs to be found. This solution is one of the possibilities under case 4.

The practical implication of this result is that it reduces the number of combinations that need to be examined. Cases 1,2 and 3 correspond to unique solutions and case 4 provides one solution for each element of each of the two sets. The total number of solutions is $L = (|S| - 1) + (|D| - 1) + 3 = |S| + |D| + 1$. If S and D are ordered, the algorithm needed for finding the optimal solution is of linear time complexity and does not, in average, enumerate all solutions, see Alg. 4.4. The initial assumption is that case 3 is the optimal one and then case 4 is examined, individually on each set. Cases 1 and 2 result naturally at the end of the iteration on each set. Minimality of the subsets is not an assumption, but it reduces the number of nodes that have to be processed.[6]

The problem becomes more complex when left and right specialisation is taken into account. As mentioned above, the set of siblings is separated into five mutually exclusive subsets, (L_s, R_s, L_d, R_d, Q). Nodes which are excluded by others can be ignored here, too. Thus, the assumption is that the five sets are independent and minimal. The non-excludable nodes can also be ignored, as before. Nodes which can only be excluded by one of the two parameters have also a minor effect on the search. Namely, they set a minimum pruning cost for this parameter, which needs to be taken into account in the search.

The discussion here will concentrate on the remaining nodes, i.e., those which can be pruned by either of the two parameters and are not excluded by others. As an example, assume the following subsets of siblings:

$$L_s = \{(a, 10, l, 11, l), (b, 8, l, 15, l), (c, 3, l, 7, r)\}, \qquad R_s = \{(d, 7, r, 9, l), (e, 5, r, 12, l)\}$$

$$L_d = \{(f, 11, l, 8, l), (g, 12, r, 7, l)\}, \qquad R_d = \{(h, 13, l, 6, r)\}$$

The first important property of this problem is that the two composite sets $S = L_s \bigcup R_s$ and $D = L_d \bigcup R_d$, have similar independence properties as before. Thus, all nodes in S, have

[6]The process involves 4 pre-processing steps: splitting the original set into S and D, sorting each of S and D, selecting the minimal and independent subsets of S and D. The most expensive step is the sorting. All others have a linear complexity, like the ORAsib search.

Input: Two ordered, independent sets of siblings S and D.
Output: The optimal pruning choice (n_i, n_j).
ORAsib(S, D) **:**

```
p* ← (head(S), head(D))                    % initial optimal solution
c* ← cost^a(p*)
c1 ← dur-cost^b(head(S))
WHILE (c1 < c* ∧ S ≠ {}):
        n2 ← head(S)                       % node in S to be pruned by duration
        c2 ← c1
        REPEAT
                S\{head(S)}
                c2 ← dur-cost(head(S))     % duration cost of next node in S
        UNTIL (c2 > c1 ∨ S = {})           % higher duration cost or end of set reached
        IF S ≠ {}:
                p3 ← (head(S), n2)
                c3 ← cost(p3)
                IF c3 < c*: p* ← p3 ∧ c* ← c3 % update optimal solution
        ELSEIF c1 < c*: p* ← p2 ∧ c* ← c1   % update optimal solution
        ENDIF
ENDWHILE
% Repeat the process for D, updating the optimal solution p*
RETURN p*
```

[a] Calculate solution cost.
[b] Duration-pruning cost for one node.

Algorithm 4.4: Description of the search algorithm for optimally pruning a set of sibling nodes.

lower distance-pruning costs than their equivalent in D and vice versa. However, it is no longer the case that *all* the nodes in D are pruned when a node in S is pruned by duration. It is possible, for example, that none of the nodes in S are pruned right by duration. Therefore, by pruning the nodes in S by duration, the R_d nodes remain unpruned. This effect has to be taken into account by the search algorithm, which needs to guarantee that all the nodes in both sets are pruned. In the above example, this phenomenon does not occur. S contains both nodes which can be pruned left by duration, e.g. a, and nodes which can be pruned right by duration, e.g. c. Similarly for D.

Nodes in a left specialisation set, e.g. L_s, cannot be excluded by nodes in the corresponding right specialisation set, R_s, by the preferred parameter, distance in this case. In other words, pruning a by distance does not, by definition, have any effect on d, e, which need to pruned in a different way. However, pruning a by duration will cause the pruning of d, because they

can both be pruned left by duration and $d(b) = 9 < d(a) = 11$. For this reason, the preferred choice for pruning *all* nodes in S, may differ from the combination of the preferred choices for pruning L_s and R_s individually. For example, the solutions $(a, d, -, -)$ and $(c, -, b, -)$, which cost $10 + 7 = 17$ and $3 + 15 = 18$ respectively, would both prune all nodes in S and the latter is only marginally more expensive than the former. Moreover, $(c, -, b, -)$ has a larger overall effect, pruning also all of the nodes in L_d, while $(a, d, -, -)$ cannot have any effect on the nodes of D, due to the independence properties of S and D.

The above examples illustrate the kinds of dependencies, which have to be taken into account during the search. The most important problem is to find the optimal solution for pruning all the nodes of S and D independently. The number of combinations which have to be considered for set S are $(|L_s| + 1) \times (|R_s| + 1)$ and similarly for D. A simple algorithm for enumerating these solutions has a quadratic time complexity. Table 4.3 presents the solutions generated and evaluated by this algorithm for set S in the above example.

Generated solution	Cost	Preferred solution	Cost
(a, d, f, h)	31	(a, d, f, h)	31
(b, d, a, h)	32	(a, d, f, h)	31
(c, d, b, h)	31	(a, d, f, h)	31
$(-, d, b, c)$	29	$(-, d, b, c)$	29
(a, e, d, h)	30	$(-, d, b, c)$	29
(b, e, a, h)	30	$(-, d, b, c)$	29
(c, e, b, h)	29	$(-, d, b, c)$	29
$(-, e, b, c)$	27	$(-, e, b, c)$	27
$(a, -, e, h)$	28	$(-, e, b, c)$	27
$(b, -, e, h)$	26	$(b, -, e, h)$	26
$(c, -, b, h)$	24	$(c, -, b, h)$	24
$(-, -, b, c)$	22	$(-, -, b, c)$	22

Table 4.3: Trace of the extended ORAsib algorithm, looking for an optimal pruning choice on four independent sibling subsets.

The algorithm simply lists all possible combinations of the nodes in L_s and R_s, attempting to prune them first by their preferred parameter and then by the alternative. The best choice in this case is $(-, -, b, c)$, which prunes all nodes by duration. A similar process takes place for the D set, generating $3 \times 2 - 1 = 5$ solutions. However, the optimal overall choice remains $(-, -, b, c)$ in this case. This approach is less efficient compared to the one for the simpler

problem, presented in Alg. 4.4, because it needs to enumerate all combinations for each of the two sets S and D. However, it still makes use of the independence properties between S and D, achieving reasonable time complexity.

4.6 Model initialisation and cost functions

One aspect of the proposed refinement method which has not been paid enough attention so far is the choice of cost functions. Since the presented algorithms guarantee the optimal solution, the choice of optimality criterion is the most important determinant of the performance of the method. The generalisation, c_g, and specialisation, c_s, cost functions used so far take a very simplistic approach. Namely the minimisation of the change incurred on the original models. This feature makes them prone to the initialisation of the most specific and most general models for the parameters.

To give an extreme example, assume that the initial MGG_d for a subevent A is $MGG_d(A) = [4..10]$ and its MGG_s, relative to B, $MGG_s(A, B) = [-40..40]$. Now assume the following negative example: $A(0, 5), B(35, 40)$. The preferred specialisation change in this case would be to make $MGG_d(A) = [6..10]$, rather than $MGG_s(A, B) = [-40..29]$. However, this is unlikely to be the desired refinement solution. Thus, it seems reasonable to scale the cost function by the initial size of the refined range.

An additional issue which has not been examined is how the initial MGG and LGG ranges are set. These are necessary for the calculation of the refinement cost. A reasonable heuristic approach for the LGG range would be to select the midpoint of the PPS range. Thus if the parameter range provided by the expert is $PPS_d(A) = [5..9]$, then $LGG_d(A) = [7..7]$ seems like a reasonable starting point. This choice is in accordance to the minimum change bias for the original model. A similar approach might be followed for the MGG model, i.e., extending the PPS range equally on both sides, e.g. $MGG_d(A) = [2..12]$. Alternatively, the expert may be required to provide these values. Thus, the assumption continues to be that the initial most specific and most general models are provided for each event in the TCN.

Based on this assumption an attempt is made here to choose a cost function which provides a good estimate of the real refinement cost. To simplify the discussion the following models

are assumed:

$$LGG = [0..lg], PPS = [0..ps], MGG = [0..mg],$$

where $lg \leq ps \leq mg$ and only right specialisation and generalisation is possible, i.e., the low bound of 0 is fixed. Assuming also a positive parameter value x, the following are some features which the generalisation and specialisation cost functions should possess:

1. The extent of change incurred to the corresponding model should be measured, i.e., $x - lg$ for generalisation and $mg - (x - 1)$ for specialisation. A dual of this criterion is the generality of the resulting model, i.e., $mg - x$ for generalisation and $(x - 1) - lg$ for specialisation. For a given $mg - lg$ the two criteria are equivalent.
2. The effect on the preferred model, if there is any, should be taken into account, i.e., $x - ps$ for generalisation and $ps - (x + 1)$ for specialisation.
3. The cost value should be scaled in a way that makes it comparable for different parameters in the model, i.e., where $mg - lg$ might differ.

The functions examined so far satisfy only the first of these requirements.

A simple function which achieves all three requirements for generalisation is the following:

$$c_g = \begin{cases} \frac{x-lg}{mg-lg+1}, & \text{if } x \leq ps \text{ or} \\ \frac{x-lg}{mg-lg+1} + \frac{x-ps}{mg-ps+1}, & \text{if } x > ps. \end{cases}$$

The basic function, $\frac{x-lg}{mg-lg+1}$, is a scaled version of the one used in the chapter so far. Moreover, it adds an extra penalty, $\frac{x-ps}{mg-ps+1}$, if the PPS is affected. Similarly for specialisation:

$$c_s = \begin{cases} \frac{mg-(x-1)}{mg-lg+1}, & \text{if } x > ps \text{ or} \\ \frac{mg-(x-1)}{mg-lg+1}, + \frac{ps-(x-1)}{ps-lg+1} & \text{if } x \leq ps. \end{cases}$$

4.7 Summary and critique

The task of refining a TCN event recognition model differs from standard knowledge refinement tasks in several respects. In particular, the construction of examples from a stream of training data presents an interesting problem. In this chapter, the concept of an event support tree has been introduced for dealing with this problem. The EST is a means of associating a number of subevent sequences, which share a common terminal subevent occurrence. This is of interest, due to the direct correspondence of terminal subevents to the defined events, as

was seen in chapter 3. For the task of parameter refinement, an incremental method has been proposed, which uses a limited memory structure. This structure holds a most general and a most specific model for each event definition, which serve as an upper and lower bound for the actual parameter settings. The two models are specialised and generalised respectively, in response to positive and negative examples. Optimal generalisation and specialisation algorithms have been introduced, which make use of the EST structure to avoid exhaustive enumeration, in the average case. Finally, the choice of optimality criterion for the two algorithms has been discussed.

The refinement method presented in this chapter suffers from two important problems: order-dependence and noise-sensitivity. The first problem results from the incremental nature of the algorithm and the restricted memory structure. An alternative would be to maintain a number of most specific and most general models, rather than just a single pair. In other words, perform a beam search over time, rather than a greedy one. A different approach to the extension of the memory structure is presented in the next chapter. The problem of noise-sensitivity is a result of the optimality requirement for the refinement search. If there is noise in the data, it is possible that a suboptimal, according to the data, solution should be sought for. A heuristic search method, which takes this problem into account, is also presented in the chapter 5.

Another problem, which becomes more important in the extended-memory structure in chapter 5 is the size of the generated ESTs, especially the negative ones. One way in which this can be restricted is by the use of the MGG models, to discard for example subevents which have occurred a long time ago. This restriction is necessary for a realistic refinement system. A different way to reduce the memory requirements is to use the event support network as a representation, which eliminates some of the duplication in the EST. However, the design of search algorithms on an ESN seems problematic. This approach is not pursued in this thesis.

Chapter 5

Lazy refinement under full supervision

The aim in this chapter is to develop a refinement method which is independent of the presentation order of the training data and tolerates noise. The target of the refinement is the same as in the previous chapter, i.e., refinement of the parameters in an event definition, given a sequence of subevent occurrences and the event occurrences that should be recognised. In order to achieve order independence, the memory structure is extended to store more information about the training data. This structure is updated incrementally and is used by a *lazy refinement* algorithm, which provides a new approach to the parameter refinement task. The method also deals with the issue of noise in the data, allowing for contradictions and imperfect classification in the training set. The heuristics used for the refinement search combine a model-based (*proximity*) with a data-based (*purity*) cost function.

5.1 Order independence

The refinement algorithms presented in chapter 4 perform a search for the optimal changes to the maximal (MGG) and minimal (LGG) parameter settings of a given event recognition model, with a knock-on effect on the preferred settings (PPS). This search uses information about the current MGG and LGG settings and the latest positive or negative example of an event. The optimal changes which are generated are adopted and the parameter settings are modified accordingly.

The main problem with this incremental approach is that the solutions generated are only "locally" optimal in time. Assume for example the following event definition:

`IF` $A(i_A)$ `AND duration`(Z, A, d_A)

`AND` $B(i_B)$ `AND duration`(Z, B, d_B) `AND precedes`(Z, A, B, s_{AB})

`THEN` $Z(i_Z)$ `WHERE` $i_Z^- = \min(\{i_A^-, i_B^-\})$ `AND` $i_Z^+ = i_B^+$

and the parameter settings:

$$\begin{array}{lllllll} PPS(Z) = PPS_d(A) = [2..4] & \wedge & PPS_s(A,B) = [0..3] & \wedge & PPS_d(B) = [1..2], \\ LGG(Z) = LGG_d(A) = [3..3] & \wedge & LGG_s(A,B) = [2..2] & \wedge & LGG_d(B) = [1..1], \\ MGG(Z) = MGG_d(A) = [1..5] & \wedge & MGG_s(A,B) = [0..5] & \wedge & MGG_d(B) = [1..5]. \end{array}$$

Assume also that the desired parameter ranges are:

$$PPS(Z) = PPS_d(A) = [2..4] \wedge PPS_s(A,B) = [0..2] \wedge PPS_d(B) = [1..4].$$

The following sequence of subevent occurrences:

$$\begin{array}{l} A(0,2) \\ B(2,3) \rightarrow Z(0,3) \\ A(4,7) \\ B(10,14)\ (\nrightarrow Z(4,14)) \\ A(15,18) \\ B(20,24) \rightarrow Z(15,24), \end{array}$$

corresponds to two positive examples:

$$(d_A = 2, s_{AB} = 0, d_B = 1) \wedge (d_A = 3, s_{AB} = 2, d_B = 4)$$

and several negative ones, out of which only one satisfies the MGG ranges and is of interest:

$$(d_A = 3, s_{AB} = 3, d_B = 4).$$

The first positive example leads to the expansion of the LGG ranges:

$$LGG_d(A) = [2..3];\ LGG_s(A,B) = [0..2];\ LGG_d(B) = [1..1]$$

and the first negative example causes the retraction of the MGG range for the duration of B:

$$MGG_d(B) = [1..3],$$

which excludes the negative sequence ending with $B(10,14)$. This decision is locally optimal, since it does not involve changes to the PPS ranges as all other candidate solutions would have done. However, this choice makes it impossible to cover the second positive example, which requires that the duration of B is 4.

This situation would not have occurred if the examples were encountered in a different order. For example, the sequence:

$A(0,2)$

$B(2,3) \rightarrow Z(0,3)$

$A(4,7)$

$B(9,13) \rightarrow Z(4,13)$

$A(14,17)$

$B(20,24)$ $(\nrightarrow Z(14,24))$

corresponds to the same positive and negative examples as above. In this case, the first two positive examples lead to the following LGG parameter settings:

$$LGG_d(A) = [2..3];\ LGG_s(A,B) = [0..2];\ LGG_d(B) = [1..4]$$

and the negative one, leads to the correct retraction of the MGG for the distance between the two subevents:

$$MGG_s(A,B) = [0..2],$$

which is the only way to exclude the negative sequence.

The problem of order dependence is common to all incremental learning and refinement algorithms that make locally optimal decisions. It has been identified and studied in almost all sub-domains of empirical learning.[1] The standard solution to the problem is to store the accumulated training information in some form of memory structure. The commonly used memory structures can be classified into two general categories: *data memory* and *model memory*.

Data memory. Under this approach the training data set, or part of it, is stored and can be used to make a global decision on the basis of the accumulated information. The methods using this approach are usually batch processing algorithms modified to solve incremental refinement tasks. Examples of these are the ID4 [91] and ID5R [101] algorithms for the incremental construction of decision trees. The latter is an improved version of the former and has the interesting property of incorporating the storage of the examples with the classification model. This is achieved by extending the decision tree representation to store frequency histograms for the examined features, plus the parts of the examples that are not covered by the current best-choice decision tree.[2]

[1] It is considered for example throughout [50], which is an elementary machine learning textbook.

[2] This is needed, since the decision tree allows classification under partial feature information.

Model memory. An alternative approach is to record the parts of the search space explicitly considered at each incremental step. A method which adopts this approach is the candidate elimination algorithm [64], which was described briefly in chapter 4. This algorithm uses the version space memory structure which maintains all most general and most specific models, that are incrementally constructed. In the context of the methods presented in chapter 4 this means that all candidate changes to the minimal and maximal models are stored at each local decision point.

The latter approach is less commonly used, due to its high storage cost. Search space explosion is common in real learning and refinement problems and it is difficult to find efficient ways of storing all possible choices. On the other hand, this method is truly incremental and does not require the re-computation or modification of the best-choice classification model. Concerns about space inefficiency have led to the selection of the former of the two approaches here. Section 5.3 describes a memory structure which extracts and stores useful information from the training examples, allowing order-independent refinement of the event recognition models.

5.2 Noise tolerance

The second major assumption that is made in chapter 4 is that the training data are correct. This is an unrealistic assumption for real-world problems, where uncertainty in data acquisition is unavoidable. For instance, noise could be introduced in the example above in the form of a mismeasurement of the time stamp for an event. Thus, if a small measurement error in the third B occurrence had occurred, the following sequence could be received:

$$
\begin{aligned}
&A(0,2)\\
&B(2,3) \rightarrow Z(0,3)\\
&A(4,7)\\
&B(9,13) \rightarrow Z(4,13)\\
&A(14,17)\\
&B(\mathbf{19},\mathbf{23})\ (\nrightarrow Z(14,23))
\end{aligned}
$$

and it would not be possible to exclude the negative example of Z. The problem arises due to the contradiction between the second positive and the negative example. They both use

subevent occurrences of the same duration and the same distance between them. Despite that, their classification is different, i.e., one of them leads to the recognition of Z, while the other does not.

The situation described above is only one of the possible types of noise which could be encountered. Types of noise which are expected to be commonly encountered are the following:

- Input or feedback measurement error.
- Missing event in input or feedback.
- Extra event in input or feedback.

Each of these situations leads to mislabelling of examples, which possibly contradicts correctly labelled ones, as shown in the example above. The approach adopted here for such problems, is to tolerate misclassification. The refinement algorithm no longer makes the assumption that all the examples are correctly labelled and allows for imperfect classification of the training data. There still is a bias for good classification performance, which is enforced by a fitness function called *purity*, but the absolute maximum of this function is no longer required.

Assuming the selection of an appropriate purity function, it is expected that the refined model will be a robust solution to the problem, despite the misclassification of some of the training data. Statistically, this is more likely to be the case, the larger the proportion of the search space which is explored, i.e., the larger and more diverse the training set that is used. This contradicts the assumption of refinement with a small training set. In order to resolve this contradiction, an additional model-based fitness function is used, called *proximity*, which introduces a bias for minimum model modification. The combined *fitness* function, using purity and proximity, guides the refinement search to solutions which achieve good classification performance and are close to the original model.

The usual approach to noise handling in empirical machine learning is different from the one adopted here. The difference stems from the fact that in empirical learning there is no restriction to the size of the training set and no initial model. In that situation, the trade-off is not between classification performance and model change, but between classification performance and model complexity. The underlying assumption is that simpler models are more robust, i.e., more likely to model the process generating the data. Examples of this work is the post- and pre-pruning of decision trees [9, 83, 69].

5.3 Extended memory structure

In chapter 4, the concept of Event Support Trees (EST) was introduced to represent positive and negative examples of an event. Each branch of the tree, from root to leaf, represents a sequence of subevent occurrences which could cause the recognition of the higher-order event. Positive ESTs hold positive examples of the event and negative ones store negative examples. Thus, the set of all ESTs generated over time for an event constitute the training set for that event. The memory structure presented in this section utilises this observation to allow order-independent parameter refinement.

5.3.1 Relative event support trees

The extended memory structure presented here belongs to the category termed *data memory* in section 5.1, since it accumulates information about the examples presented to the system up to a certain point in time. The *model memory* alternative has been rejected, due to the large number of model changes that satisfy individual examples. As seen in chapter 4, in the case of negative examples the search space is highly exponential and storing all of the alternative model changes would cause an exponential increase in space requirements. Thus, storing the training data and performing batch refinement is preferable. However, the explicit storage of all ESTs is also inefficient, since each individual EST can be quite large. For this reason, the memory structure translates the ESTs into a more compact representation which is also more suitable for parameter refinement.

The new memory structure is called a *Relative Event Support Tree* (REST) and extends the EST representation in several ways. The first major difference from an EST is that the nodes do not hold absolute, but relative time information. Instead of a time stamp, each node of a REST holds a `(distance,duration)` pair, which corresponds to the duration of the subevent occurrence in the EST and its distance from the subevent occurrence that it precedes, i.e., its parent node in the EST. Figure 5.1 illustrates how a REST is built by the set of ESTs for the first example examined in section 5.1. Strictly speaking, a REST is not a tree, but a *forest*, i.e., a set of trees, each corresponding to a different duration for the terminal subevent in the definition. However, it is convenient to think of it as a tree, containing a dummy node which relates the individual trees. Note also that the distance information for the nodes of the terminal subevent is undefined, since it does not precede another subevent.

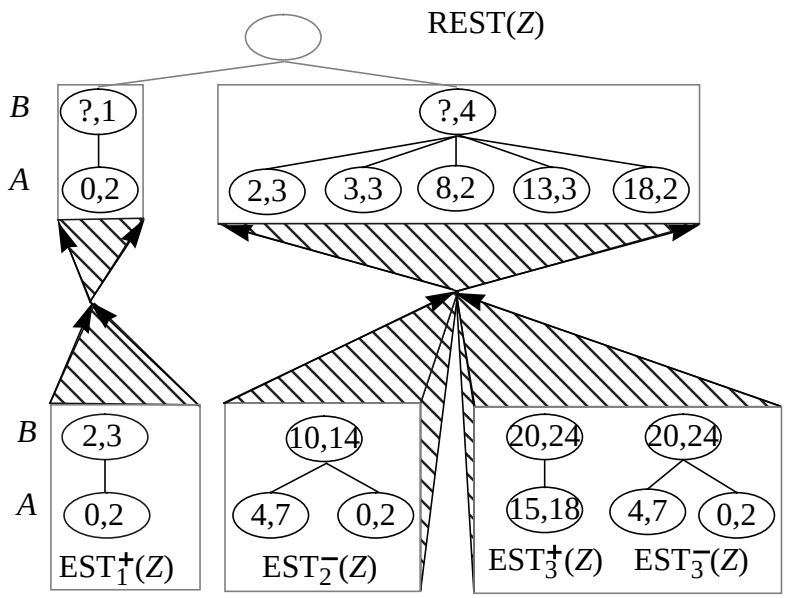


Figure 5.1: From EST to REST. The index of the EST corresponds to the example number and the superscript to whether it is a positive (+) or a negative (-) one.

The REST can take advantage of the repetition in the parameter values encountered in the examples to reduce its space requirements. An example of this phenomenon in Fig. 5.1 is the node $(?, 4)$ for subevent B. This node corresponds to three different occurrences of the subevent. Moreover, the use of relative time removes the need for a transformation from the time domain to the parameter space, where the refinement search takes place. The search can now be done on the basis of the values held in the nodes of the REST. For example, by setting the duration range for B to $PPS_d(B) = [1..4]$, both nodes of the terminal subevent in REST(Z) are covered, since they hold the duration values: $d_B = 1$, $d_B = 4$. Therefore, unless excluded by the remaining PPS ranges, i.e., $PPS_s(A, B)$ and $PPS_d(A)$, all examples of Z will be covered by this choice.

The information in the REST of Fig. 5.1 is not sufficient for deciding about the coverage of each model. In addition to the relative representation of the examples, the REST structure needs to hold information about the example sequences that each node participates in. For instance, the node $(?, 4)$ for B in Fig. 5.1, corresponds to one positive and two negative examples of Z. This information is needed, in order to calculate the purity of a refinement solution. The usual approach to this problem, e.g. [101], is to store frequency counts of the encountered

examples. However, frequency information is not sufficient for RESTs, due to the phenomenon of alternative positive examples. As explained in chapter 4, a positive EST represents a set of event sequences, of which one is sufficient for the satisfaction of the example. Therefore, if two nodes at the same level of the REST are derived from the same positive EST, covering one of them is as good as covering both. This information cannot be deduced by frequency counts alone. Each positive example covered by a REST node must be uniquely identifiable. The solution adopted for this problem is to use a global example counter, which is used as a unique label for each EST. This number corresponds to the index of the ESTs in Fig. 5.1. Thus, each REST node can hold a list of the positive example labels that it covers. This is not necessary for negative examples, since all sequences in each negative EST need to be excluded. Therefore a simple count of the negative examples covered by each REST node suffices.

Figure 5.2 illustrates the local storage of positive and negative examples in the REST of Fig. 5.1. The square brackets beside each REST node contain a set of labels for positive examples, enclosed in braces, and the number of negatives covered. In this case, the positive sequences do not have alternatives and therefore each positive example number appears only once in each level of the REST.

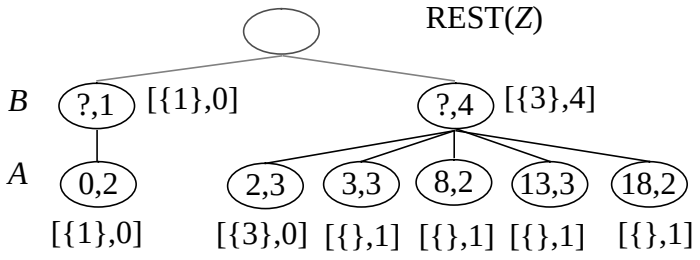


Figure 5.2: A REST with coverage information.

5.3.2 Pruning the relative event support tree

The size of the REST can increase substantially, by considering nodes which correspond to parameter values far outside the range of interest in the search. The usual cause of this problem is the large size of negative ESTs. The identification and pruning of nodes which are not of interest increases both the space and time efficiency of the refinement algorithm. The obvious problematic situation is the consideration of very large distance values. The A node $(18, 2)$

of the REST in Fig. 5.2 is an example of this situation. Since the original parameter range for the distance between A and B is $[0..3]$ and small changes to it are required, this node with a distance of 18 is very unlikely to affect the search for new distance parameters. Similar situations can occur for the minimum distance and the minimum and maximum duration limits. This is especially so in the following situations:

1. The classification network is of high order, i.e., it contains event definitions of high order. High-order events are usually of longer duration than lower-order ones, since the duration of each conjunctive event is always at least as long as the longest subevent. In this situation, introducing boundaries to the duration ranges of interest is useful.
2. There is a high degree of overlap between the defined events in the network. In that case, large negative distances can be recorded, which again are of no interest.

The solution adopted for this problem is to introduce *windows* of interest for each parameter range. These windows are ranges, similar to the MGG ranges, used in chapter 4. For example, the window for $PPS_s(A, B) = [0..3]$ might be $MGG_s(A, B) = [-3..6]$. If that was the case, no distance values outside the MGG range would be considered in the construction of the REST. The specification of parameter windows can be done either manually or automatically. For the automatic specification of a window, the original range is extended in proportion to its own size. The function in (5.1) is used to calculate the expansion on each side of a range.

$$\textit{expansion}([l^-..l^+]) = \max(b \times (l^+ - l^-), m) \tag{5.1}$$

where $[l^-..l^+]$ is the original range, e.g. $PPS_s(A, B) = [0..3]$, b is a factor of change and m is a minimum bound to the required expansion. The parameters b and m are manually defined. The minimum expansion bound is needed, in order to avoid problems when the original range is very small. The window for $[l^-..l^+]$ is then:

$$[(l^- - \textit{expansion}([l^-..l^+]))..(l^+ + \textit{expansion}([l^-..l^+]))].$$

If the examined range is a duration one, the lowest duration limit must be ≥ 1.

To illustrate the calculation of a window, consider the expansion of the distance range $PPS_s(A, B) = [0..3]$, under the assumption that $b = 1$ and $m = 2$:

$$\textit{expansion}(0, 3) = \max(1 \times 3, 2) = 3.$$

The corresponding parameter window is $MGG_s(A, B) = [-3..6]$, which prunes the REST in Fig. 5.2 to that of Fig. 5.3. The new REST focuses on the most relevant event sequences.

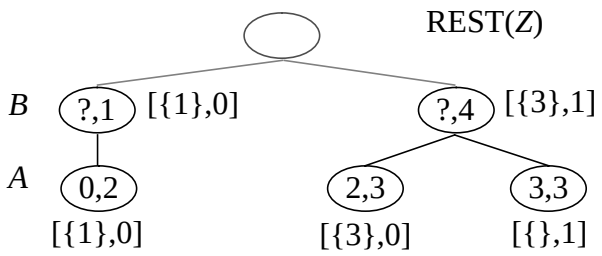


Figure 5.3: Example of a pruned REST.

5.4 Refinement under full supervision

The REST structure facilitates the storage of positive and negative examples for each event, under the assumption that training feedback is provided for each of the events in the TCN. This mode of refinement is termed *full supervision*. The accumulated information can be used at any point in time to refine the parameters of the model. Clearly, as more training data are received and encoded in the RESTs, the refinement choice is more informed and therefore more robust. This section describes a refinement algorithm which is invoked by the user of the system, whenever refinement is desired. Due to its inactivity, during the incremental accumulation of information, this type of algorithm is called a *lazy* one. The common characteristic of lazy refinement and learning algorithms is the use of a data, rather than a model, memory.

The lazy refinement algorithm (LRA), uses the REST of an event to perform a heuristic search for the correct parameter settings in the definition of the event. This search takes place at two levels:

- A *generate and test* algorithm (LRAloc) performs a local search at a single level of the REST. The result of this search provides the duration and distance ranges for one of the subevents in the definition.
- LRA then performs a *best-first search*, selecting and combining the results of individual local searches, to construct a complete solution.

5.4.1 Local search for new parameters

The local search algorithm focuses on one of the subevents in the definition, proposing the refinement of the corresponding distance and duration ranges. The algorithm searches this

two-dimensional space for rectangular regions, which cover as many positive and exclude as many negative examples as possible. Each REST node is represented by a point in the space, which can be positive and/or negative, depending on whether the node participates in positive and/or negative sequences. Figure 5.4 presents the search space for the nodes corresponding to subevent A in the unpruned REST of Fig. 5.2. The task here is simple: the problematic negative point at $(3, 3)$ can be excluded by a small retraction of the parameter range.

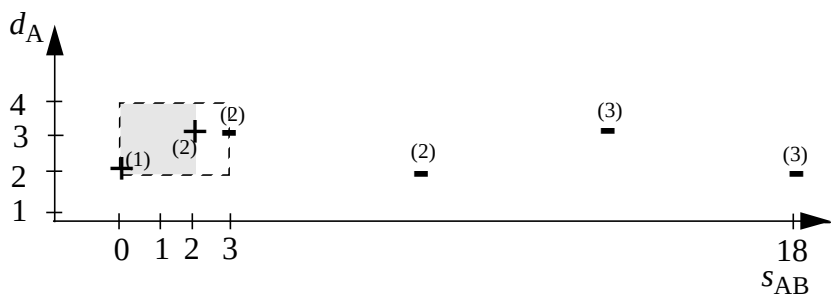


Figure 5.4: The space for local search, under full supervision. The dashed rectangular region corresponds to the original parameters and the shaded region to the desired ones. Plus points correspond to positive examples and minus ones to negative. The example label is attached in brackets.

LRAloc generates alternative parameter settings and evaluates their fitness with respect to:

- their purity, i.e., their ability to distinguish between positive and negative points, and
- their proximity to the original parameter settings.

The purity measure introduces a bias for correct classification, i.e., a *data bias*, and the proximity represents a bias for minimum change, i.e., a *model bias*. The two measures are combined into a fitness function. In order to describe the algorithm, the following generic fitness function will be used:

$$f([s^- .. s^+], [d^- .. d^+]) : \mathbb{Z}^2 \times \mathbb{Z}^2 \rightarrow [0..1], \tag{5.2}$$

which provides a real value between 0 and 1 for each *local solution*, comprising the ranges $[s^- .. s^+]$ for distance and $[d^- .. d^+]$ for duration. Such a *local solution* is represented here by the following notation (e, s^-, s^+, d^-, d^+), where e is the subevent which is examined. Local solutions with high proximity and purity scores have a high fitness, i.e., close to 1. The choice of a fitness function is examined in section 5.5.

A naive exhaustive approach to the problem would generate *all* possible combinations of duration and distance ranges and evaluate them. In the above example, such a method would need to consider:

- 19 alternatives, all values in the range $[0..18]$, for the lower distance limit;
- for each of these, all distance values above it, for the upper limit;
- and similarly for the two duration parameters.

Even in this simple example, an exhaustive search would examine $190 \times 3 = 570$ candidate solutions.

The first simple improvement to this approach is to ignore solutions that cover exactly the same set of points. In other words, the parameter values of the *given* points can be used to generate the candidate solutions. In doing so, care has to be taken to select values that are as close as possible to the original ones, since the goal is to change the original settings as little as possible. For example, the solution which covers only the leftmost positive example would be $(A, 0, 1, 2, 4)$ instead of $(A, 0, 0, 2, 2)$. This method reduces the number of candidates to 21.

This argument can be taken further by defining situations where a candidate solution is *subsumed* by another. As an example consider the solutions $(A, 0, 3, 2, 4)$ and $(A, 0, 8, 2, 4)$. The latter one is clearly further away from the original parameters and covers the same points as the former, plus one negative. In this situation, the latter candidate is said to be subsumed by the former. The following set of criteria determines whether a solution h_1 subsumes another solution h_2:

- h_1 has a higher proximity than h_2 and
- it covers all positive points that h_2 covers and
- it does not cover more negative points.

With the use of subsumption, all solutions generated by the three rightmost negative points in Fig. 5.4 can be ignored. The resulting set of candidates contains only two solutions: $(A, 0, 2, 2, 4)$ and $(A, 0, 3, 2, 4)$. Note that the second solution corresponds to the original parameter settings and is retained, i.e., it is not subsumed by the former, because of its high proximity value.

The search, as described so far, maintains the optimality of the exhaustive search, but makes use of constraints, which allow it to concentrate on an interesting subset of the complete set of solutions. With the use of this method, the size of the search space can be significantly reduced, although not always as much as in the simple example above. In the worst case, the complexity

of the search is quartic to the number of training data points. Although this is just a worst case estimate, it does cause concern about the efficiency of the search in handling complex examples.

Taking into account the fact that the local search needs to be performed several times in the course of the search for a complete solution, it is necessary to improve the efficiency of the local search algorithm further. This improvement is achieved by abandoning the optimality requirement and replacing the exhaustive with a greedy heuristic search. The heuristic search transforms the two-dimensional search problem into three one-dimensional ones, as described below:

- Assuming the most general duration range necessary to cover all data points, select the best solution on the distance axis, according to the fitness function f which incorporates both purity and proximity information. In the example of Fig. 5.4, this stage would produce the solution:

  $$(A, 0, 2, ?, ?),$$

  where the $?$ sign denotes that the duration parameter is not relevant at that stage. In addition, the solution $(A, 0, 3, ?, ?)$ is generated, but not expanded further, because it is assumed to have a lower fitness than $(A, 0, 2, ?, ?)$. $(A, 0, 3, ?, ?)$ is closer to the original distance range $PPS_s(A, B) = [0..3]$ and therefore has a higher proximity than $(A, 0, 2, ?, ?)$. However, it covers a negative example and has low purity.
- Do the same for the duration axis, resulting in:

  $$(A, ?, ?, 2, 4).$$

- Select the best of the two solutions, e.g. distance refinement:

  $$(A, 0, 2, ?, ?),$$

  and, using only the points covered by this solution, perform again the one-dimensional search on the other axis, e.g. duration. The result in this case is:

  $$(A, 0, 2, 2, 4).$$

Clearly, this algorithm will not always produce the best overall solution. One of the rejected solutions may perform better than the chosen one, when the second dimension is considered.

However, the complexity of the greedy algorithm is only quadratic and by considering the best solution on *both* dimensions individually before constraining the search, a close approximation of the optimal solution is expected in most cases. A detailed specification of the local search algorithm is given in appendix D.2.

5.4.2 Combining local changes

A *complete solution* consists of a set of local solutions, one for each subevent. Thus, in addition to the local search algorithm, a method for moving between levels of the REST and combining the results is needed. This task is achieved by a best-first search method, which performs a top-down traversal of the REST, applying the local search algorithm to each level visited. The best-first search algorithm, constructs candidate solutions incrementally and stores them in a set of solutions, which for the most part will be *partial*, i.e., incomplete. The search stops when the best solution is a complete one.

Figure 5.5 illustrates the search performed by the algorithm, using the simple REST of Fig. 5.3. At the top-level of nodes, i.e., subevent B, the algorithm generates two solutions:

$$(B, ?, ?, 1, 2),$$
$$(B, ?, ?, 1, 4),$$

constraining the duration of B. The first one covers one positive example and the second all three examples, i.e., two positive and a negative. The proximity of $(B, ?, ?, 1, 2)$ is higher than that of $(B, ?, ?, 1, 4)$, because the original duration range for B is: $PPS_d(B) = [1..2]$. Suppose that the second candidate has a larger overall fitness value and is selected to be expanded. Since it covers all nodes at the top level, all nodes at the lower level have to be considered. At that level, the local search works as explained in section 5.4.1, updating the set of solutions to:

$$\begin{aligned}
&(B, ?, ?, 1, 2), \\
&(B, ?, ?, 1, 4) \quad \wedge \quad (A, 0, 3, ?, ?), \\
&(B, ?, ?, 1, 4) \quad \wedge \quad (A, ?, ?, 2, 4), \\
&(B, ?, ?, 1, 4) \quad \wedge \quad (A, 0, 2, 2, 4).
\end{aligned}$$

The above is the set of complete and partial solutions at the end of the refinement search. Note that both of the solutions generated in the first stage of the local search , i.e., $(A, 0, 3, ?, ?)$ and $(A, 0, 2, ?, ?)$, are retained. However, the latter, is expanded further to give $(A, 0, 2, 2, 4)$.

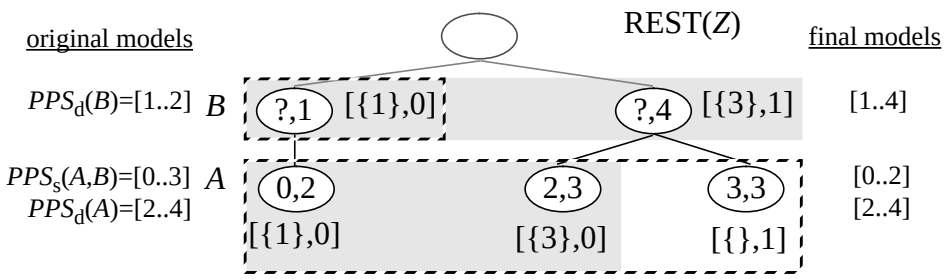


Figure 5.5: Best-first search, using the REST. The dashed rectangles correspond to the rejected local solutions, while the shaded ones to the chosen solution.

If the last of the above solutions is chosen, it will be returned as the best overall solution, because it is a complete one. An interesting alternative path would have been to expand the partial solution $(B, ?, ?, 1, 2)$, instead of $(B, ?, ?, 1, 4)$. The best overall solution would then have been:

$$(B, ?, ?, 1, 2) \ \wedge \ (A, 0, 3, 2, 4),$$

which has a higher proximity and excludes the negative example, but fails to cover one positive example. No change to the parameters of A is needed, because $(B, ?, ?, 1, 2)$ prunes the negative example at the level of B. Algorithm 5.1 gives a narrative description of the LRA algorithm. A more detailed specification is given in appendix D.3.

An important feature of the algorithm is the way in which it incrementally builds the candidate solutions. It *expands* a partial solution, examining the children of the nodes that it covers. The children are treated as if they were siblings, although they are children of different REST nodes. The only requirement is that all the real sequences of subevents covered by the parent nodes are also covered by the children set. This is warranted by the direct correspondence of the REST to EST nodes, i.e., the fact that each EST node is mapped onto a unique REST node.

The LRA algorithm does not always yield the optimal result. Its performance is dependent on the choice of the fitness function used in the evaluation of the generated solutions. Its main advantage is that it uses the REST in an efficient way. It uses the inherent relational structure of the REST, i.e., the dependence of the children nodes to their parent, in order to focus on the relevant areas as the search progresses. This is realised by the top-down traversal of the REST. Also, by making the local search independent from the global best-first search, the complexity

Input: The REST for an event.

Output: The `best-solution` parameters for the event model.

LRA(REST**):**

1. Initialise `solutions` and `best-solution` to the empty set.
2. Set the `current-level` to be the top level of the REST, i.e., the nodes corresponding to the terminal subevent.[a]
3. Call LRAloc, to perform a local search using the `current-level` nodes. Sort the resulting solutions by their fitness value. The set of generated solutions includes the solutions generated on all three stages of the local search, even those which constrain only one of the two parameter ranges, i.e., the ones generated in the first two stages of the local search.
4. Generate a new *partial* solution from each local one, by adding the local solution to the `best-solution`, according to fitness, i.e., proximity and purity. This should be the head of the `solutions` list.
5. Insert the new solutions to the `solutions` set, maintaining the order by fitness. Store the `current-level` with each solution.
6. Choose the new `best-solution`.
7. `IF` the `best-solution` contains unconstrained parameter values:
   - perform the third stage of the local search,
   - replace it with the new set of solutions and repeat 6 and 7.

   `OTHERWISE IF` the `best-solution` is not complete:
   - construct the set of children of the nodes covered by the solution,
   - set the `current-level` to the level stored with the solution,
   - remove the selected solution and perform 3 to 7 on the constructed children set.

   `ELSE` return the `best-solution`.

[a] Note that at the top level the search is one-dimensional, since the distance parameters are undefined.

Algorithm 5.1: The best-first global search algorithm (LRA).

of the multi-dimensional global refinement task is reduced in a natural way.

5.5 Cost functions

Due to the greedy nature of the search, the choice of the fitness function plays a very important role in the overall performance of the refinement method. The small size of the training set means that the heuristic has to make the most of the available data. For this reason model-based information is also taken into account. The way in which this is done has a significant effect on performance. This section examines various alternatives for the heuristic and presents their merits and shortcomings. There are certainly many more functions that could be used and the final choice will be in many cases application-dependent. The goal here is to give some

indication of the form that the heuristics need to take and their desirable properties.

The fitness function, introduced in (5.2), consists of two independent measures for the quality of a local solution:

- *purity* evaluates the classification performance of the solution, i.e., the proportion of positive and negative examples that it covers. The generic form for this function is as follows:

$$\textit{purity}((s^-, s^+, d^-, d^+)) : \mathbb{Z}^4 \rightarrow [0..1], \tag{5.3}$$

  where (e, s^-, s^+, d^-, d^+) is the local solution, e being the subevent signature. A simple purity function is the proportion of positive examples covered by the solution. Applying this function to the local solution $(A, 0, 2, 2, 4)$ of section 5.4.1, the result is:

$$\textit{purity}((0, 2, 2, 4)) = 1,$$

  since the solution covers both of the two positive examples.

- *proximity* calculates the amount of change to the original parameters and has a similar form:

$$\textit{proximity}((s^-, s^+, d^-, d^+)) : \mathbb{Z}^4 \rightarrow [0..1]. \tag{5.4}$$

  Given, for example, the initial parameter ranges:

$$PPS_s(A, B) = [0..3] \wedge PPS_d(A) = [2..4],$$

  the proximity of the local solution $(A, 0, 3, 2, 4)$ is:

$$\textit{proximity}((0, 3, 2, 4)) = 1,$$

  because it does not change the parameter ranges.

In addition to the functions defined for local solutions, their counterparts for sets of local solutions, i.e., partial and complete solutions, need to be defined. These *global* functions evaluate sets of solution quadruples and take the following form:

$$\textit{Purity}(S) : \mathcal{P}(\mathbb{Z}^4) \rightarrow [0..1], \tag{5.5}$$

$$\textit{Proximity}(S) : \mathcal{P}(\mathbb{Z}^4) \rightarrow [0..1], \tag{5.6}$$

where S is a set of local solutions and $\mathcal{P}(\mathbb{Z}^4)$ the power set of local solution quadruples.

In order to facilitate the comparison of the measures presented in the next sections, the following are some important issues regarding the choice of fitness function:

1. The *purity* heuristic should deal with alternative positive examples, as defined in chapter 4. For each local solution, a set of alternative positives should count as a single positive example.
2. It should be possible to give different weight to positive and negative examples in *purity*. Due to the definition of negative examples, as all non-positive sequences, the number of positive examples is expected to be smaller than that of negative ones. This raises a problem which becomes serious in the following situations:
   - defining large parameter windows will allow the inclusion of irrelevant sequences, which in most cases are negative,
   - the combination of a highly-connected TCN and overlapping events leads to the recognition of spurious sequences and a corresponding large number of negative examples.

   Since the degree to which these situations occur is not known, a flexible way to handle the unbalanced training set is needed.
3. The derivative of the *Purity* and *Proximity* functions should be of similar order. This will simplify the fitness function which combines the two functions.
4. The fitness function should address the issue of relative importance of the data and model biases, i.e., purity and proximity. Usually, this will depend on the amount of expected noise in the data and the confidence to the original parameters of the model.
5. The global *Purity* and *Proximity* functions should facilitate the comparison of solutions that make use of different subtrees in the REST. Despite their incremental calculation, they should be global in nature and evaluate solutions based on a common denominator.
6. The computation of the local functions should be cost effective.

5.5.1 Purity

The local *purity* heuristic is a measure of the correct classification for a solution. Thus, given a local solution (e, s^-, s^+, d^-, d^+), the first step is to calculate the number of positive and

negative examples that it covers. For example, the local solution $(A, 0, 2, 2, 4)$ in section 5.4.1 covers two positive examples and no negative ones, while $(A, 0, 3, 2, 4)$ covers two positive and one negative example. The way in which these numbers are calculated is by looking at the example lists in each node covered by a local solution. Assuming that M_i is the set of nodes covered by the ith solution and P_j, n_j, the set of positive and number of negative examples covered by the jth node in M_i, the number of covered positive and negative examples is:

$$p_i = |\bigcup_j P_j|,\ n_i = \sum_j n_j,$$

where $\bigcup_j P_j$ should not contain duplicates. Given the number of covered examples, and the total number of positive (p) and of negative (n) examples, a function of the following form is required, which measures the classification accuracy of the solution:

$$\textit{accuracy}(p_i, n_i, p, n) : \mathbb{N}^4 \rightarrow [0..1]. \tag{5.7}$$

This type of function has been the subject of substantial research in machine learning, resulting in numerous alternative metrics. There have also been comparative studies of the commonly used heuristics. In one of these studies [63], it is argued that the choice of heuristic does not make much difference in the overall performance of the system, since they all guide the search in a similar manner, namely they reward the correct classification of examples. This is not the case here, because of the different treatment of positive and negative examples.

A popular measure of classification accuracy is based on the minimisation of information entropy in the classification task. The argument for this heuristic is as follows (from [82]):

1. If there is no model describing the training data, the information content of the sample can be calculated using entropy:

$$E_0(p, n) = -(\frac{p}{p+n} \log \frac{p}{p+n} + \frac{n}{p+n} \log \frac{n}{p+n}). \tag{5.8}$$

2. With the introduction of a classification model, the entropy of the problem should decrease. The achieved gain is given by *gain*$= E_0 - E_i$, where E_i the new entropy for

each local solution. In the examined classification task, E_i is calculated as follows:

$$\begin{aligned}
&E_i(p_i, n_i, p, n) = \\
&\frac{p_i + n_i}{p + n} \times E_{cov}(p_i, n_i) \; + \; \frac{p + n - p_i - n_i}{p + n} \times E_{exc}(p - p_i, n - n_i) = \\
&-(\frac{p_i}{p + n} \log \frac{p_i}{p_i + n_i} + \frac{n_i}{p + n} \log \frac{n_i}{p_i + n_i}) \\
&\quad - (\frac{p - p_i}{p + n} \log \frac{p - p_i}{p + n - p_i - n_i} + \frac{n - n_i}{p + n} \log \frac{n - n_i}{p + n - p_i - n_i}),
\end{aligned} \tag{5.9}$$

where E_{cov} corresponds to the area of the instance space covered by the solution and E_{exc} to the excluded area.

3. Since E_0 is constant, the goal is to minimise E_i. Thus the function of interest is:

$$\textit{gain}(p_i, n_i, p, n) = 1 - E_i. \tag{5.10}$$

In the context of the event recognition task, the *gain* function has the following problems:

1. It is designed to maximise the discrimination between positive and negative examples and therefore rewards solutions that cover many negative examples and exclude many positive. Figure 5.6 illustrates the behaviour of the *gain* function, under the assumption that $p = n$, i.e., there are as many positive as negative examples. The problem is shown by the symmetry of the plot: maximum reward is given when all positives are covered and all negatives are excluded or vice versa.
2. It is biased towards the majority class. If the examples are not equally distributed between the two classes, the class which corresponds to the majority will have a larger impact on the heuristic. Figure 5.7 illustrates this point. It depicts a situation where there are many more negative than positive examples. Clearly, changes to the number of covered positive examples have very little effect on the purity of the solution. This problem is known and variants of the heuristic have been developed that address it (see [85]). These variants are not examined here, because they are more complex and less intuitive than the simple entropy heuristic.
3. The entropy calculations are computationally expensive, due to the logarithmic components of the function.
4. The non-linearity of the function makes its combination with the proximity heuristic difficult. A similar type of proximity function is needed to provide a sensible fitness heuristic.

For these reasons the entropy-based purity heuristic was rejected. The decisive factor was the last of its properties as listed above. The problems become more apparent in section 5.5.2, where the proximity heuristic is examined.

An alternative linear function which has a similar behaviour is the simple classification ratio, as defined in (5.11).

$$\textit{ratio}(p_i, n_i, p, n) = \frac{p_i + n - n_i}{p + n}. \tag{5.11}$$

This heuristic also appears in the machine learning literature, e.g. [50], but it is less popular due to its lack of theoretical justification. It can be seen as the quantitative expression of the intuitive goal to "maximise correct classification." It overcomes most of the practical problems that were expressed above for the entropy-based heuristic, but still suffers from a bias for the majority class. Figures 5.8 and 5.9 illustrate the behaviour of the heuristic, under the two situations examined above. The tilt of the hyper-plane, observed in Fig. 5.9 shows the effect of an unbalanced split of the examples.

Function (5.11) can be modified to remove the bias for the majority class:[3]

$$\textit{sratio}(p_i, n_i, p, n) = 0.5 \times \frac{p_i}{p} + 0.5 \times \frac{n - n_i}{n}. \tag{5.12}$$

The following equation shows the relation of the *sratio* with the *ratio* function:

$$\begin{aligned} \textit{ratio}(p_i, n_i, p, n) &= \frac{p_i + n - n_i}{p + n} \\ &= \frac{\mathbf{p}}{\mathbf{p} + \mathbf{n}} \times \frac{p_i}{p} + \frac{\mathbf{n}}{\mathbf{p} + \mathbf{n}} \times \frac{n - n_i}{n}. \end{aligned}$$

The highlighted parts of the above transformation show how the *ratio* function weighs the positive and negative classification ratios of the *sratio* function, according to the initial bias in the training set. Figures 5.10 and 5.11 illustrate how the *sratio* measure remains unaffected by the relative size of the positive and negative classes in the original data set. However, this choice is also not without problems. Imagine, for instance, the situation where the examined REST contains 100 positive and just 1 negative example. The coverage of the single negative example will be given the same importance as the coverage of the 100 positive ones. A solution which covers all positive and the negative example, will have a purity of:

$$\textit{sratio}(100, 1, 100, 1) = 0.5$$

[3] *sratio* stands for scaled ratio.

and a solution which covers a single positive, but excludes the negative will have a higher purity:

$$\textit{sratio}(1, 0, 100, 1) = 0.51.$$

The conclusion of the above discussion is that the choice of an appropriate *purity* function remains an open question. What differentiates this problem from standard approaches in empirical learning is the different status of positive and negative examples. A negative example can be excluded in a number of different ways, i.e., by pruning the negative sequence at one of the possible pruning points, while a positive one will only be covered if it satisfies all of the constraints. In conjunction with the expected large number of negative examples, this observation suggests that positive examples are more important than negative ones. However, it is difficult to quantify this phenomenon and capture it in an appropriate *purity* function. An alternative approach to the modification of the *purity* function is to identify the "important" negative examples and ignore the others. This idea is implemented by ignoring the negative examples, which:

- do not satisfy the original PPS parameter ranges and
- do not interfere with positive examples, i.e., they lie outside the hyper-rectangular area covering *all* positive examples.

This filtering of negative examples aims to balance the treatment of positive and negative examples. If this is achieved, *ratio* and *sratio* will behave similarly.

Moreover, in some cases, it may be desirable to bias the search towards one of the classes. This could happen for example in a life-critical decision task, such as a medical one, where the correct classification of positive cases is more important than that of negative ones. This is clearly an application-dependent issue and can be addressed by a user-defined parameter, which is used as a relative weight for the two classes. This parameter will be referred to as *positive-to-negative weight* and symbolised by $\beta \in [0..1]$. The augmented *ratio* and *sratio* functions, incorporating this feature are shown below.

$$\textit{ratio}(p_i, n_i, p, n) = \frac{\beta p_i + (1-\beta)(n - n_i)}{\beta p + (1-\beta)n}. \tag{5.13}$$

$$\textit{sratio}(p_i, n_i, p, n) = \beta\frac{p_i}{p} + (1-\beta)\frac{n - n_i}{n}. \tag{5.14}$$

So, the higher the value specified for β, the more important positive examples become in the evaluation of the solution.

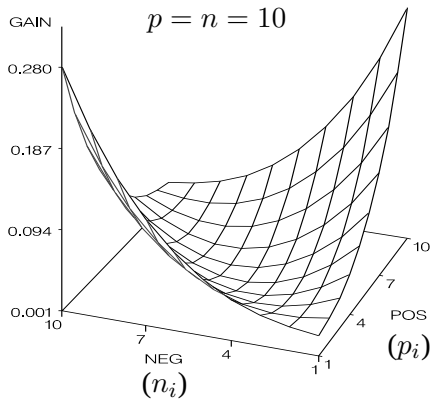


Figure 5.6: Gain ratio, balanced.

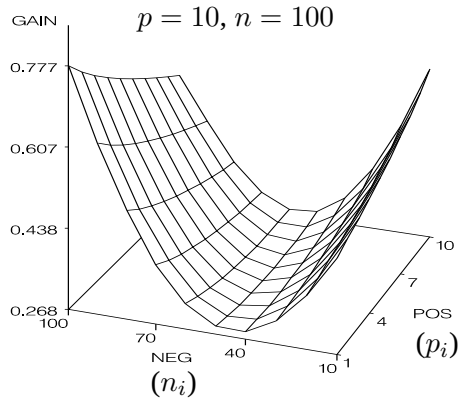


Figure 5.7: Gain ratio, unbalanced.

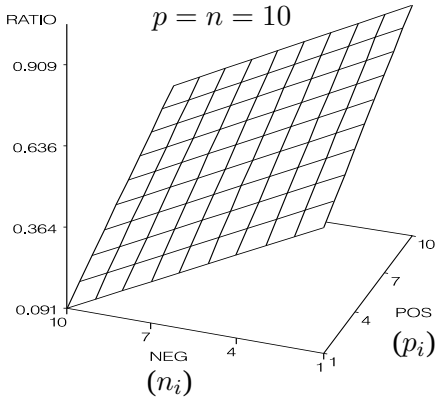


Figure 5.8: Simple classification ratio, balanced.

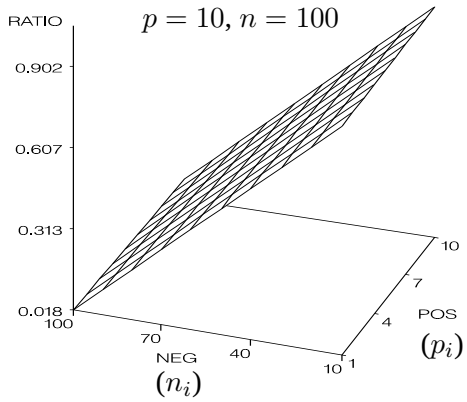


Figure 5.9: Simple classification ratio, unbalanced.

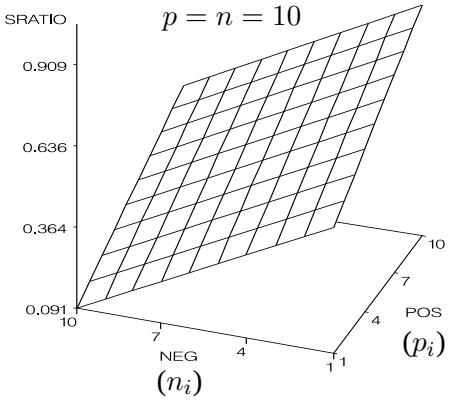


Figure 5.10: Scaled classification ratio, balanced.

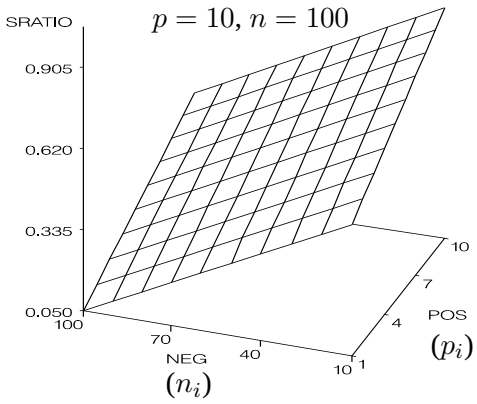


Figure 5.11: Scaled classification ratio, unbalanced.

The *Purity* function. The definition of the *Purity* function, which is used for the evaluation of sets of local solutions, i.e., partial and complete solutions, is effectively the same as for local ones. Since it evaluates a conjunction of conditions, i.e., the individual local solutions, the number of examples covered at the lowest level of the REST, corresponds to the coverage of the whole solution. For example, the complete solution:

$$(B, ?, ?, 1, 4) \wedge (A, 0, 2, 2, 4)$$

in Fig. 5.5, has the same coverage as its member local solution: $(A, 0, 2, 2, 4)$ at the level of the of the subevent A, i.e., two positive and no negative examples. Therefore its *Purity* is equal to 1. The *Purity* function is defined as follows:

$$\textit{Purity}(S) = \textit{purity}((s_n^-, s_n^+, d_n^-, d_n^+)) \tag{5.15}$$

where S is the partial or complete solution and $(s_n^-, s_n^+, d_n^-, d_n^+)$, the nth of the n local solutions in S, i.e., the lowest-level one.

5.5.2 Proximity

Proximity is a measure of "closeness" of the parameter ranges in a solution to the original settings. Thus, it should reach a maximum if the solution does not cause any changes to the model and it should decrease as the required change increases. The local *proximity* receives as input two pairs of ranges and needs to calculate their proximity to the original ranges at that level of the REST. For example, the proposed parameters for event A, by the solution in Fig. 5.5 are $[0..2]$ for distance and $[2..4]$ for duration, which need to be compared with the original settings $[0..3]$ and $[2..4]$. The global *Proximity* measure compares a set of such pairs with the equivalent set of original settings.

According to the criteria listed at the beginning of section 5.5, the proximity measure ought to take a form that allows it to be combined with the purity function. Therefore it is desirable to choose a linear function which takes values in the range $[0..1]$. A marginal change in one of the two heuristics will then be comparable to a marginal change in the other.

Assuming that the generated local solution is (e, s^-, s^+, d^-, d^+) and the corresponding original parameters for subevent e in the model are $(e, s_0^-, s_0^+, d_0^-, d_0^+)$, the local proximity heuristic should:

- measure the displacement of $[s^- .. s^+]$, with respect to $[s_0^- .. s_0^+]$,
- measure the displacement of $[d^- .. d^+]$, with respect to $[d_0^- .. d_0^+]$,
- combine the two measurements and
- scale the result to be in the range $[0..1]$.

A simple measure of the displacement of one range from the other is the sum of the differences between their limits:

$$\begin{aligned} \textit{disp}(s^-, s^+, s_0^-, s_0^+) &= |s^- - s_0^-| + |s^+ - s_0^+|. \\ \textit{disp}(d^-, d^+, d_0^-, d_0^+) &= |d^- - d_0^-| + |d^+ - d_0^+|. \end{aligned}$$

This corresponds to the city-block dissimilarity measure on two dimensions. There are many similar measures that could be used instead, e.g. Euclidean distance. The city-block is chosen due to its simplicity.

The two measurements can be combined by summation, but the difficulty is in choosing a sensible scaling factor that will produce the desired $[0..1]$ output. The solution adopted here is to use the parameter windows to calculate a maximum displacement for each parameter range. For example, if the window for the distance parameter is $[s_w^- .. s_w^+]$, the maximum displacement is:

$$\textit{max-disp}(s_0^-, s_0^+, s_w^-, s_w^+) = \max((s_0^- + s_0^+ - 2s_w^-), (2s_w^+ - s_0^- - s_0^+)).$$

One problem which can arise with this approach is that the maximum possible displacement is too large, causing the calculation of very small relative displacement for most solutions. An alternative approach is to use the maximum and minimum positive points in the data set. The simplifying assumption is made here that the two approaches provide similar results, i.e., the parameter windows are good indicators of the range of possible values for the parameter.

Combining the above ideas, the following *proximity* measure is defined:

$$\begin{aligned} &\textit{proximity}_1(s^-, s^+, d^-, d^+) = \\ &\quad 1 - \frac{\textit{disp}(s^-, s^+, s_0^-, s_0^+) + \textit{disp}(d^-, d^+, d_0^-, d_0^+)}{\textit{max-disp}(s_0^-, s_0^+, s_w^-, s_w^+) + \textit{max-disp}(d_0^-, d_0^+, d_w^-, d_w^+)}. \end{aligned} \tag{5.16}$$

Clearly the measure is linear and takes values in the range $[0..1]$, as required. It reaches the maximum value 1 when the displacement on both axes is at the minimum, i.e., the original ranges remain unaltered. One assumption which underlies the measure is that the marginal

displacement on the duration axis has the same weight as that on distance. An alternative approach would be to scale the time-unit in inverse proportion to the maximum displacement on each of the two parameters:

$$\begin{aligned} &\textit{proximity}_2(s^-, s^+, d^-, d^+) = \\ &\quad 1 - 0.5 \times \left(\frac{\textit{disp}(s^-, s^+, s_0^-, s_0^+)}{\textit{max-disp}(s_0^-, s_0^+, s_w^-, s_w^+)} + \frac{\textit{disp}(d^-, d^+, d_0^-, d_0^+)}{\textit{max-disp}(d_0^-, d_0^+, d_w^-, d_w^+)}\right). \end{aligned} \quad (5.17)$$

The relation between the two functions is similar to the relation of *ratio* and *sratio* for purity. However, there is no intuitive reason for tolerating larger changes on the dimension with the larger maximum displacement. On this basis, the former heuristic is preferred.

The *Proximity* function. The global *Proximity* of a partial or a complete solution can be defined as the average *proximity* over its component local solutions. The implicit assumption is that the position of a subevent in the event definition does not affect its weight, i.e., changes in the parameters of B are as important as changes in the parameters of A. Under this assumption, the definition of *proximity* introduced in (5.16) can be extended to the set of parameters examined in a partial or a complete solution:

$$\begin{aligned} &\textit{Proximity}(S) = \\ &\quad 1 - \frac{\sum_{i=1}^{n}(\textit{disp}(s_i^-, s_i^+, s_{i0}^-, s_{i0}^+) + \textit{disp}(d_i^-, d_i^+, d_{i0}^-, d_{i0}^+))}{\sum_{i=1}^{n}(\textit{max-disp}(s_{i0}^-, s_{i0}^+, s_{iw}^-, s_{iw}^+) + \textit{max-disp}(d_{i0}^-, d_{i0}^+, d_{iw}^-, d_{iw}^+))}, \end{aligned} \quad (5.18)$$

where S is a set of n local solutions. The assumption of uniform treatment of unit changes is extended here to the complete set of parameters.

Alternative proximity measures

A number of alternative measures to the displacement ratio have been considered as candidates for the *proximity* measure. A common criticism for these alternatives is that they are more complex than the displacement ratio.

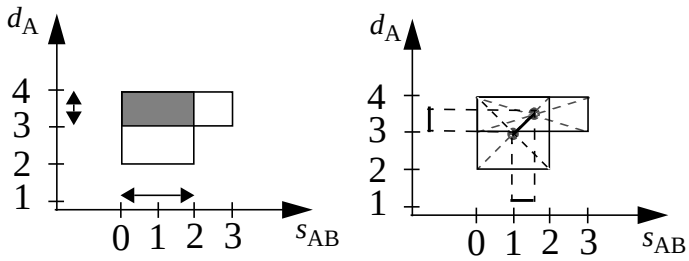
Overlap ratio. This metric is an attempt to measure the part of the search space covered in common by the original and the new parameters. Clearly, the higher this overlap, the closer the new parameters are to the old ones. Two variants of this approach have been considered, that are listed below.

1. One-dimensional overlap, i.e., individual treatment of the overlap on each dimension.

2. Two-dimensional overlap, i.e., the overlap of the rectangles corresponding to the parameter quadruples. In effect this is the product of the one-dimensional duration and distance overlap.

Both these measures can be scaled between 0 and 1 with the use of the maximum overlap, which is simply the original range or rectangle. Figure 5.12(a) illustrates schematically the two measures. In two dimensions, the overlap is the shaded rectangular area. Its one-dimensional variant is shown as the projection of this area on the two axes. One problem with the overlap measures is that they provide no way of measuring the distance between two non-overlapping regions. Moreover, they differentiate between expansion and retraction of the original parameter ranges. For example, any range that covers completely the original one has maximum overlap with it.

Distance of centroids. This measure is inspired by the common geometric method for measuring distances between objects. It involves the calculation of the Euclidean distance between the centroids of the two objects. Figure 5.12(b) illustrates how this measure works. In two dimensions, the distance is shown as the line connecting the centroids of the two rectangles. Its one-dimensional variant is shown as the projection of the two-dimensional distance on the two axes. For similar reasons to the overlap metrics, this measure does not provide an adequate solution to the problem. It is possible for example, that the new and the old parameter ranges share the same centroid, without being identical.



(a) Overlap of ranges. (b) Distance of centroids.

Figure 5.12: Alternative proximity measures.

5.5.3 Combining purity and proximity

In the two previous sections simple model-based and data-based heuristics were introduced, which evaluate the quality of a solution. In order to construct a measure for the overall fitness of the solution, the two heuristics need to be combined in a meaningful way. Two approaches for the combination of *Purity* and *Proximity* were considered: the weighted average and the product of the two measures:

$$f_1(S) = \gamma \text{Purity}(S) + (1-\gamma)\textit{Proximity}(S) \tag{5.19}$$

$$f_2(S) = \text{Purity}(S) \times \textit{Proximity}(S) \tag{5.20}$$

The parameter γ is application-dependent and it is meant to be a measure of the quality of the data, relative to the confidence of the expert in the original model. It takes values in $[0..1]$ and it is equal to 0.5 when the data and the model ought to be given the same weight. In the case of noisy data, it should take values below 0.5, while in the case of high uncertainty in the original parameter settings, γ should be above 0.5.

The multiplicative approach has the property of favouring solutions that have a moderate *Proximity* and *Purity* value, in comparison to solutions which do very well in one of the two heuristics. As an example, consider a solution S_1, evaluated as follows:

$$\textit{Purity}(S_1) = 0.6 \ \& \ \textit{Proximity}(S_1) = 0.4$$

and a second one S_2, for which:

$$\textit{Purity}(S_2) = 1.0 \ \& \ \textit{Proximity}(S_2) = 0.2.$$

The combined fitness value will be:

$$f_2(S_1) = 0.24 \ \& \ f_2(S_2) = 0.2$$

and S_1 will be chosen, although it does not do very well according to either of the two heuristics. An additional property of this approach is that it cannot discriminate between solutions for which *Proximity*$= 0$ or *Purity*$= 0$.

The weighted summation, on the other hand, provides a linear combination of the two heuristics, which gives more intuitive results. In the above example, assuming that $\gamma = 0.5$ it

would give:

$$f_1(S_1) = 0.5 \ \& \ f_1(S_2) = 0.6$$

leading to S_2 being preferred over S_1.

5.6 Refining repeating events

The main focus of this chapter has been on the refinement of temporal constraints in conjunctive event definitions. Disjunctive definitions and repeating events have so far been ignored. This section diverges from the main route to examine other event types. In particular, a restricted solution to the refinement of temporal constraints in repeating event definitions is presented. The solution is limited by a number of simplifying assumptions, aiming to reduce the size of the search space. Disjunctive events, on the other hand, can be dealt with naturally by the proposed method, treating each of the alternative terminal subevents individually and creating a separate REST for each.

5.6.1 Search space for repeating events

The definition of repeating events contains three pairs of numeric parameters: the range of legal durations for the supporting event, the range of legal distances between repetitions and the range which restricts the number of repetitions. The last of the three numeric ranges is non-temporal and will not be dealt with here. Thus, the first assumption is that the minimum and maximum number of repetitions is fixed manually.

Despite the small number of refinable parameters in the definition of a repeating event, the representation of example subevent sequences with the use of Event Support Trees (ESTs) has a high combinatorial complexity. For example, assume that event A^* is defined as 2 to 4 repetitions of event A:

$$A \times [2..4] \rightarrow A^*.$$

Assume also that the following sequence appears in the input data:

$$A(0,2), A(3,4), A(6,7), A(8,9).$$

Even in this simple case, there are 11 different sequences which could lead to the recognition of A^*. These are represented by the three ESTs in Fig. 5.13, of size: 2, 4 and 8 nodes respectively. In general, given a sequence of n occurrences of A and the range of repetitions $[n^-..n^+]$, the number of example subevent sequences for A^* is given by:

$$\sum_{i=n^-}^{n^+} \binom{n}{i},$$

which corresponds to the number of unordered subsets of size n^- to n^+ of the set of A occurrences.[4]

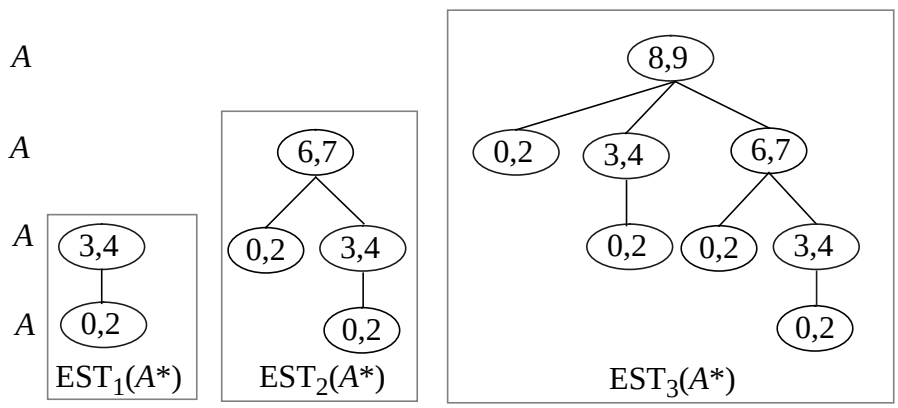


Figure 5.13: ESTs for an example of a repeating event. The first two ESTs are temporally subsumed by the third. Note that each branch of length $> n^- = 2$ corresponds to more than one example of A^*.

The following are some interesting properties of the ESTs in Fig. 5.13:

1. They do not have a uniform depth. All levels of the EST correspond to the single supporter, A, of the repeating event, A^*. The length of different branches varies, as the number of repetitions of A required for the recognition of A^* is not a single number, but a range.
2. The first EST is a subtree of the second, which is in turn a subtree of the large one. Thus, the occurrences of A^* corresponding to the first two trees, i.e., $A^*(0,4)$, $A^*(0,7)$ and $A^*(3,7)$, are temporally subsumed by the occurrences in the last tree. Nevertheless, the large tree does not represent these occurrences, since it uses $A(8,9)$ as the terminal subevent occurrence.

[4]This result is obtained under the assumption that no parameter windows are used.

3. Some of the branches correspond to multiple occurrences of A^*, e.g. $A^*(6,9)$, $A^*(3,9)$ and $A^*(0,9)$ are all represented by the rightmost branch of the third EST. These occurrences differ only in their begin time point.
4. There are only 6 occurrences of A^*, covering all of the 11 sequences of A. Moreover, all of these occurrences are subsumed by the longest one: $A^*(0,9)$.

The relative event support tree (REST), which summarises the three ESTs is shown in Fig. 5.14. Due to its direct correspondence to the underlying ESTs, the REST inherits all of the above properties.

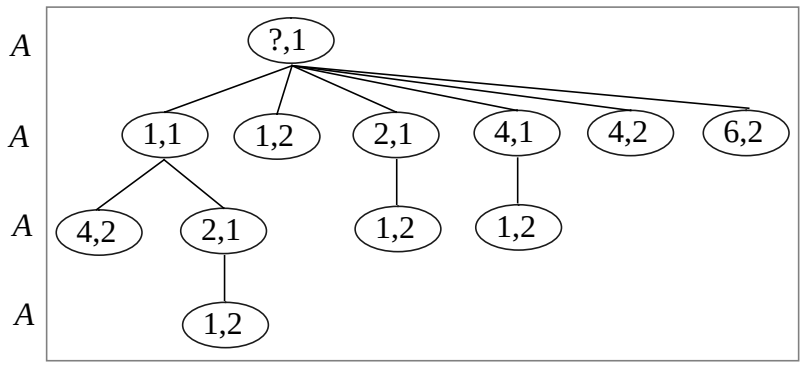


Figure 5.14: REST for a repeating event. Each branch of this REST corresponds to an EST branch in Fig. 5.13, e.g. $\{(?,1),(1,2)\}$ corresponds to $\{(3,4),(0,2)\}$.

It becomes clear from the above discussion that the examples of repeating events and their RESTs require a different treatment from that of conjunctive events. For example, if $A^*(0,9)$ is labelled positive, all of the 5 other potential occurrences of A^* will need to be treated as positive, because they are subsumed by $A^*(0,9)$. On the other hand, a negative example sequence may subsume a positive one, e.g. $A^*(0,9)$ may be negative, but $A^*(0,7)$ positive. The issues of sequence subsumption and variable branch length are addressed in section 5.6.3, which describes the required pre-processing of RESTs for repeating events.

5.6.2 Simplifying assumptions

In order to reduce the size of the RESTs and the search space for new parameter settings, a number of simplifying assumptions regarding the use of repeating events need to be made.

Unbroken sequences. One assumption which seems natural and reduces significantly the size of the RESTs is that the sequences of repetitions which are considered in the construction of the RESTs are unbroken, i.e., no subevent occurrence can be skipped. In the working example this means that:

1. the number of possible examples for A^* is reduced to the 6 distinct sequences and
2. each of the three ESTs has a branching factor of 1, as seen in Fig. 5.15.

This assumption leads to problems if an occurrence of A^* overlaps with one of another event which is also supported by A, or when noisy occurrences of A are recognised. In those cases the inability to ignore the interfering occurrences of A may result in the over-generalisation of distance and duration ranges.

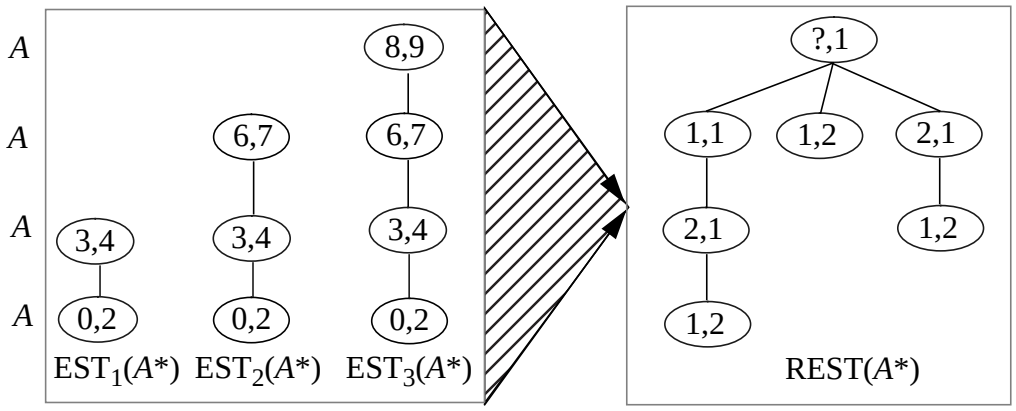


Figure 5.15: Reduced ESTs and REST for a repeating event.

Number of repetitions. The minimum and maximum number of repetitions are assumed non-refinable, as mentioned above. Moreover, the semantics of the maximum of the range need to be modified with respect to chapter 3. The maximum number of repetitions is used as a limiting, rather than a rejecting condition, i.e., sequences of a larger number of repetitions than the upper limit are not generated at all. In the working example above, sequences of 5 occurrences of A or more are not generated. However, all of their subsequences of length ≥ 2 are examined. For example, assume that the fifth occurrence of A in the input data is $A(11, 12)$. The occurrence $A^*(0, 12)$ will not be generated, but all of the ones it subsumes will be, including $A^*(8, 12), A^*(6, 12)$ and $A^*(3, 12)$. In contrast, the maximum number of

repetitions in chapter 3 was used to reject occurrences of A^*. Therefore if *all five* occurrences of A satisfied the temporal constraints in the definition of event A^*, none of the above subsequences of $A^*(0, 12)$ would be considered. The modified treatment of the repetition bound in refinement is necessary, because the real duration and distance ranges are not known a priori, and therefore it is not known whether a sequence of five A occurrences is a legal one. If, in the above example, the duration of A is constrained to be < 2, then $A(0, 2)$ will not satisfy it and the occurrences $A^*(8, 12)$, $A^*(6, 12)$ and $A^*(3, 12)$ will become legal. Proper treatment according to the original semantics of chapter 3, requires the use of the maximum number of repetitions as an alternative to the specialisation of the temporal ranges for the exclusion of example sequences. This introduces an additional degree of freedom to the search, which is of different nature from the temporal parameters.

Extended proximity bias. This last assumption states that when an example sequence is subsumed by another and both of them satisfy the original duration and distance ranges, the subsumed one is ignored. It aims to reduce the number of subsumed positive sequences, which are generated as separate occurrences of A^*.

5.6.3 Building and pre-processing the relative event support tree.

The construction of the REST for a repeating event is very similar to that for a conjunctive one: the terminal event is first placed at the root level of the tree and then the subtrees are updated by recursively moving backwards through the repeating sequence. The following are some minor differences:

- Unbroken sequences: when looking for preceding occurrences in the history of the subevent, only the most recent one can be used. For example, when looking for occurrences that precede $A(8, 9)$ there is only one option: $A(6, 7)$.
- Maximum and minimum branch length. These are imposed by the upper and lower bounds in the number of repetitions.
- Positive examples subsumed by negative. A REST node which contains only positive examples may have negative children.
- Storage of negative examples. The labels of negative examples need to be stored, as

already done for the positive ones. This is needed in order to establish the subsumption of examples as explained below.

The new element in the refinement of repeating events is the pre-processing of the REST, which precedes the search for new parameter settings. This new process deals with the peculiarities of repeating events, described in section 5.6.1. In order to explain how this is achieved, assume that $A^*(3,9)$ should be recognised from the sequence of A occurrences in the working example. Figure 5.16 augments the REST in Fig. 5.15 with the corresponding example information.

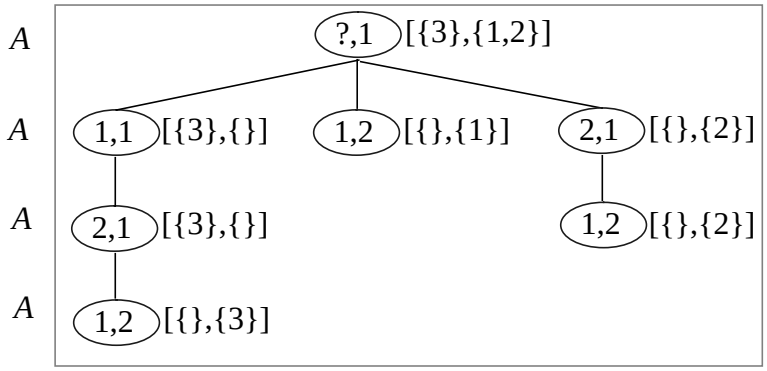


Figure 5.16: The REST for a repeating event, including positive and negative examples. The first set of numbers in braces are the positive and the second the negative examples, covered by each node.

The first important difference between repeating and conjunctive events is the temporal subsumption of example sequences. There are four possible types of this phenomenon:

1. Positive examples subsuming negative ones. This situation occurs when events in the middle of a positive sequence are used as terminal ones. This can be seen in the first two examples, 1 and 2, of A^* in Fig. 5.16. These negative examples must not be excluded, because the positive example subsuming them will be excluded, too. Thus, they should be ignored.
2. Negative examples subsuming positive ones. This situation occurs when the earlier events in a sequence are not part of the repeating event. This is the case with occurrence $A(0,2)$ in the working example. The positive and negative parts of the sequence need to be treated separately, i.e., the negative example should be excluded

in a way that does not exclude the positive.

3. Negative examples subsuming negative ones. This happens in every non-positive sequence which is longer than the minimum number of repetitions. In this case, the shortest sequence is the most important one, since its exclusion implies the exclusion of all the ones which subsume it. At first, example 2 for A^*, i.e., the rightmost branch of the REST in Fig. 5.16, seems to provide an example of this case, but the subsumed negative event turns out to be subsumed also by the positive sequence and should therefore be ignored.
4. Positive events subsuming positive ones. This happens, when the feedback data do not specify the start point of the repeating event, e.g. $A^*(?, 9)$. In this case the shortest positive example, i.e., $A^*(6, 9)$, needs to be covered, but the longer alternatives, i.e., $A^*(3, 9)$ and $A^*(0, 9)$, do not need to be excluded.

Figure 5.17 illustrates the effect of this first stage of pre-processing on the REST of Fig. 5.16.[5] The negative examples which are subsumed by positive ones are relabelled by the corresponding positive label. This applies, for example, to $A^*(3, 7)$, REST branch $\{(?, 1), (2, 1)\}$, which is subsumed by $A^*(3, 9)$, REST branch $\{(?, 1), (1, 1), (2, 1)\}$.

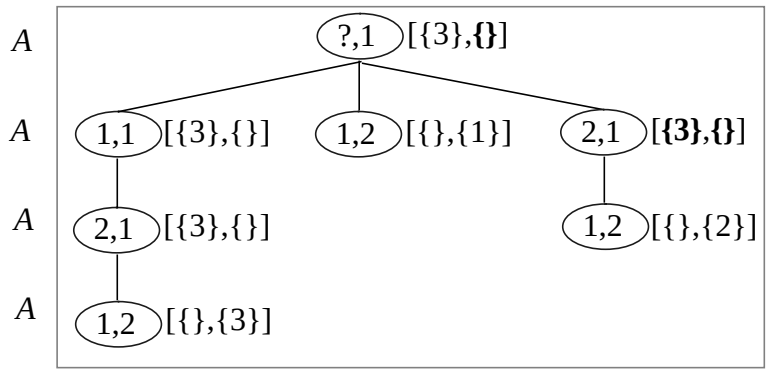


Figure 5.17: The REST without the subsumed examples. Note the relabelling of subsumed negative examples 1 and 2. The sets of examples that have changed are highlighted.

In the second stage of the pre-processing, the REST of a repeating event collapses to a single-level, "*dummy*" REST. This is possible, because there are only two pairs of temporal

[5]One technical problem which needs to be noted is that the REST itself does not contain enough information to determine all types of sequence subsumption. The histories of event occurrences need to be consulted.

parameters in the definition of a repeating event, i.e., the same duration and distance ranges apply to all levels of the REST. Thus, all of the examples in the REST can be represented by sets of points in the two-dimensional distance/duration space. Figure 5.18 illustrates the effect of this process on the REST of Fig. 5.17. In order to cover an example, the rectangle that covers all points in the corresponding set needs to be covered. In Fig. 5.17, the positive example, requires that the duration of A is $d_A \in [1..1]$ and the distance, $s_{AA} \in [1..2]$, covering the distance and duration values in the REST branch $\{(?,1),(1,1),(2,1)\}$. The three negative examples are all mapped to the same point in the space, i.e., $d_A = 2 \wedge s_{AA} = 1$, because they correspond to the same occurrence of A, i.e., $A(0,2)$. Consequently, the goal of the search for the new parameter settings, in terms of purity, is to cover the positive sets of points and exclude at least one point from each of the negative sets. By excluding one point from each negative set, the corresponding negative rectangle cannot be covered and therefore the example is excluded. In this case, the negative examples are single-point ones and the obvious solution is to exclude occurrences of A, with duration $d_A > 1$.

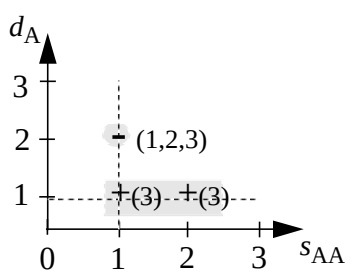


Figure 5.18: The two-dimensional representation of the REST of a repeating event. One positive example is mapped to the rectangle $(1,2,1,1)$ and three negative examples are all mapped to the same single-point area $(1,1,2,2)$.

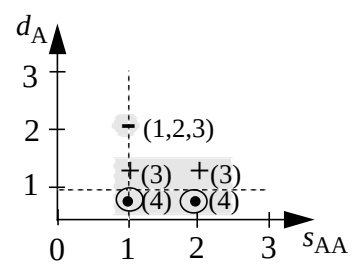


Figure 5.19: The search space for the repeating event, including an imaginary single-repetition example of duration 1.

The treatment of examples as sets of points, rather than single ones is the only difference of this representation to the 2-dimensional space used in the local search for conjunctive events, see section 5.4.1. Thus, at the end of the second stage of the pre-processing, the refinement of the temporal parameters of a repeating event definition reduces to a variant of the local search for conjunctive events. A technical problem in the collapse of the REST to a dummy one

appears when the minimum number of repetitions is 1. In that case, a single occurrence of the event A may constitute an example for A^*. Since this occurrence will not be preceded, there is no distance information available. Such an example can be represented in the 2-dimensional distance/duration space by a set of *alternative* single-point examples along the distance axis.[6] Figure 5.19 illustrates this for an imaginary example with duration $d = 1$. In order to cover this example, one of the alternative points needs to be covered. In order to exclude it, the corresponding duration value needs to be excluded.

5.6.4 Refining the temporal parameters

The basic search algorithm for new parameters in a repeating event definition is similar to the local search for conjunctive events. Their only difference stems from the fact that the examples are no longer points, but rectangles in the distance/duration space. Only a minor modification to LRAloc is needed to handle this difference. When LRAloc generates the candidate local solutions on one of the two dimensions, e.g. distance, it performs an almost exhaustive enumeration of the possible ranges on that axis, discarding some uninteresting ones as explained in section 5.4.1. This exhaustive enumeration is done in two stages:

1. Set the low range bound r^- to the parameter value for the next positive example.
2. Generate all useful ranges, with this low bound.

This process is performed for all parameter values which can serve as low range bounds. For repeating events, the low bound is set according to the low bounds of the positive examples, which are now themselves ranges. Once the low bound is set, all examples with a lower low bound are ignored and the upper bounds of the remaining examples are used to construct the candidate ranges.

For example, in Fig. 5.18 the first low bound for s_{AA} is $s^-_{AA} = 1$. Once this is set, only the upper bounds of the examples are considered. In this case, there is just one positive example, the upper bound of which is 2. Therefore, the upper bound of the only local solution is set to $s^+_{AA} = 2$. Similarly, if example 4 is positive in Fig. 5.19, the first lower bound is $s^-_{AA} = 1$ and there are two possible upper bounds: $s^+_{AA} = 1$ and $s^+_{AA} = 2$, generating two local solutions. Then the low bound is set to $s^-_{AA} = 2$ and there is only one upper bound, $s^+_{AA} = 2$, to be considered.

[6] Note that the number of these single-point examples is determined by the number of all possible distance values in the REST.

This procedure ignores the effect of the original parameter range, which is the same as for conjunctive events, i.e., candidate ranges which do not cover additional positive examples and may cover some negative ones are generated, because their limits are close to the original range. The evaluation of the generated candidate solutions is also identical to that in conjunctive events. For each solution which is generated, the number of positive and negative examples that are covered determine its purity, while its proximity score is calculated with the same proximity function. The two are combined, as usual, to provide the candidate's overall fitness.

5.7 Summary and critique

In this chapter, the task of providing order-independent and noise-tolerant refinement of the temporal constraints of a TCN model has been examined. First the concept of a *Relative Event Support Tree*, used to store information about the observed examples, was presented. A REST encodes the set of training examples for an event in a tree structure which captures the relational dependence between the subevents of each example sequence. Then a refinement algorithm was presented, which uses the REST to perform a greedy heuristic search for good parameter settings. The quality of each generated solution is evaluated in terms of its classification performance on the observed examples and its proximity to the original model. The desirable properties of the heuristic functions have been examined and two simple functions were chosen and combined into a global fitness heuristic. The final section of this chapter examined the memory structure for repeating events. The REST for this type of event differs significantly from conjunctive events, but it can be pre-processed and collapsed into a single-level REST, on which the local refinement algorithm can be applied.

A potential problem with the REST memory structure is that it can get quite large, through the duplication of nodes at the lower levels of the tree. The source of this problem is the incomplete description of the training set. Each positive or negative example may correspond to more than one subevent sequence. An EST – and its extended version, the REST – stores the complete set of possibilities in a tree. One way to limit the amount of node duplication is to replace the tree with a graph. As discussed in chapter 4, an EST is equivalent to an event support network (ESN). The ESN avoids the duplication of nodes, by representing each event occurrence in the EST by a single node. Similarly, one could use a Relative ESN (RESN),

instead of the REST, in order to accumulate information over time. The RESN would avoid the duplication of `(distance,duration)` pairs found in a REST. However, the use of a graph, rather than a tree, memory structure complicates the design of the search algorithm for refinement. For example, each child of a node in the RESN does not necessarily cover a subset of the examples covered by its parent, since it is allowed to have more than one parents. This violates one of the main assumptions made by the LRAloc algorithm.

Another important problem is the brittleness of the refinement algorithm. If the algorithm makes the wrong choice near the root of the tree, it is unlikely to find the correct solution, although it stores all the candidate solutions at each level. The source of this problem is the heuristic fitness function which favours longer, i.e., more complete solutions. In fact, the *Purity* of a partial solution never decreases with the addition of a local solution and *Proximity* will only decrease if a parameter range far from the original range is chosen.

This problem occurs, for example, in the search performed on the REST of Fig. 5.5. At the B level, two local solutions are generated:

$$q_1 = (B, ?, ?, 1, 2) \ \& \ q_2 = (B, ?, ?, 1, 4).$$

The purity of the two solutions is:

$$\textit{purity}(q_1) = 0.75 \ \& \ \textit{purity}(q_2) = 0.5$$

and, assuming that the parameter window for the duration of B is $[1..4]$, the proximity of the solutions is:

$$\textit{proximity}(q_1) = 1.0 \ \& \ \textit{proximity}(q_2) = 0.6.$$

Therefore, q_1 is chosen, and since its only child does not cause any change to the parameters for A, q_2 is not examined any further. The chosen complete solution is:

$$PPS_d(A) = [2..3] \ \wedge \ PPS_d(B) = [1..2] \ \wedge \ PPS_s(A, B) = [0..3]$$

This solution is not optimal in terms of classification performance, since it does not cover one of the two positive sequences. If q_2 was chosen instead both of the positive examples would be covered, as explained in section 5.4.2. So the overall solution might have been a better one.

The brittleness of the search algorithm can be overcome by changing the fitness function and/or changing the algorithm. The fitness function could be replaced by one that decreases

with the solution size. This would turn the search to an optimal branch-and-bound one, with respect to the chosen fitness function. The cost to be paid is an increase in computational complexity, which can be high for a realistic REST. Alternatively, the algorithm could be modified to examine all levels of the REST, before making a decision. In order to avoid a large computational cost, this modification could be combined with the replacement of the best-first algorithm with a hill-climbing one. The resulting algorithm would still perform a heuristic search, but it would make more informed local choices. The advantages and disadvantages of this alternative need to be examined further.

The computational performance of the local search algorithm could also be improved, by a parametric estimation method. For example, one could assume a Gaussian distribution of positive examples within each parameter range. The estimation of the mean and the variance for this distribution can be done in linear rather than quadratic time, which is the complexity of the LRAloc algorithm. The problem with this approach is that the choice of probability distribution cannot be justified easily, especially when the size of the data set is small.

Finally, the method which was presented for the refinement of repeating event definitions provides a restricted solution and could be improved in several ways:

- There may be more suitable memory structures than the REST for repeating events. One problem with the REST is that it does not utilise the subsumption of repeating event sequences. As a result the generated RESTs are unnecessarily large, with a large number of duplicate nodes and subtrees.
- Some of the assumptions about the nature of repeating events may be too strong for some applications. Relaxing these assumptions, under the proposed memory and search schemes, will result in a large increase in the computational and storage complexity of the system.

The solution to the above two problems may be a completely different approach to the refinement of repeating events.

Chapter 6

Refinement under partial supervision

The work presented so far has assumed that full supervision is provided for refinement, i.e., training feedback for all of the events in the temporal classification network. This assumption is not practical for large networks and is relaxed in this chapter. A refinement method is presented which requires training feedback for only the output events in a temporal classification network. This mode of refinement is termed *partial supervision*. The effect of partial supervision is to increase the ambiguity in the classification of events. This problem is tackled by a system which maintains a set of *recognition beliefs* for all of the event occurrences which can be recognised in the stream of input data. A recognition belief is an estimate of how likely it is for an event to have occurred. These beliefs are initialised on the basis of the original model and updated using the training feedback which is received for the output event occurrences. A special feedback allocation procedure is used for belief updating. Finally, the refinement of the model is done locally, using the same algorithm as under full supervision.

6.1 Partial supervision

Refinement under full supervision, as was examined in chapter 5, requires training information for each node of the temporal classification network (TCN). However, the representation of any realistic event recognition model by a TCN is likely to result in a large network of considerable depth. Under these circumstances, the provision of external training feedback for all the nodes of the network is impractical, for the following reasons:

- Only a subset of the defined nodes will correspond to the events of interest. Training information for the rest of the nodes may not be available.
- It is not always possible to represent all of the events in an event recognition model by a simple logic operator and therefore a single TCN node. In more complex definitions

it is impractical to provide training information for all subcomponents. For example, assume that event Z is a conjunction of two disjunctions, i.e.,

$$\texttt{IF}\ \ (A \vee B) \wedge (C \vee D)\ \ \texttt{THEN}\ \ Z,$$

The representation of this definition in a TCN would involve two artificial subevents for the two disjuncts. It should not be necessary to provide external feedback for these events.

- As the depth of the network increases, a large number of low-order event occurrences are expected to be recognised. Assume, for example, an event A^*, defined as a repetition of A, which is *not* an input event. It is expected that a large number of occurrences of A will be recognised and it is inconvenient to provide feedback for each one.
- A related problem is that several of the recognised low-order event occurrences will not be usable for the recognition of higher-order ones, but will also not be negative. In a noisy environment, the number of such event occurrences may be large. A mechanism is needed for ignoring such occurrences.

Hence, it is desirable that the model can be refined on partial training information. This task is termed refinement under *partial supervision*.

Partially supervised refinement differs from unsupervised learning in that training information is provided at least for those events for which there are no other means of acquiring feedback, i.e., the output nodes of the TCN. Thus, the minimal requirement for supervised refinement of a TCN model is the provision of external feedback for the output nodes of the network. Feedback for intermediate nodes is advantageous to the refinement process, but should not be necessary under partial supervision. For the work presented in this chapter, the assumption is made that classification information is only available for the output nodes. Under this assumption, the main aim is to provide a method for allocating feedback to the intermediate nodes, based on their participation in the recognition of output event occurrences. The *feedback allocation* method is combined with the refinement algorithm, presented in chapter 5, to provide a method of refinement under partial supervision.

This process is illustrated by the simple third-order TCN shown in Fig. 6.1. Events A, B, C and D are input and Y is the only output event. W, X are intermediate events, for which no

training information is provided. Assume the following sequence of input event occurrences:

$$A(0,2), B(3,4), C(5,7), D(8,10),$$

which according to the original model leads to the recognition of $X(5,10)$ only. In other words, the model parameters for W do not allow the recognition of $W(0,4)$. If the feedback for Y, does not require $Y(0,10)$ to be recognised, no change to the model is needed and the recognised X event is not used. But if $Y(0,10)$ should be recognised, then

- the model parameters for W should change to allow the recognition of $W(0,4)$ and
- if $Y(0,10)$ is still not recognised, the model parameters for Y need to be modified.

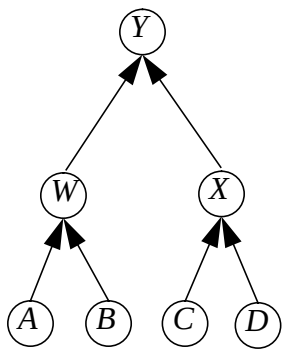


Figure 6.1: Partial supervision; a simple TCN.

Despite the oversimplification, the above example shows the main problem under partial supervision, i.e., the need to deduce the correct classification for $W(0,4)$, on the basis of its participation in the recognition of higher-order event occurrences. In a more realistic situation, the task becomes more complex through:

- complex interactions between events. For example, W, might be a subevent of another output event Z, which possibly also provides feedback about the recognition of $W(0,4)$. This may be contradictory to the feedback received from Y.
- noise in the data. The feedback received for Y may be noisy and cause undesirable changes to the models of W and Y.
- multiple solutions. There may be a choice between different occurrences of W that can cause the recognition of $Y(0,10)$.
- network of high order. Event W may be supported by non-input events, the recognition of which is also uncertain and to which it needs to allocate the feedback it

receives from Y.

All these factors increase the uncertainty in the classification of intermediate events and complicate the task of feedback allocation.

6.2 Feedback allocation and heuristic refinement

Most machine learning and knowledge refinement methods are designed to operate under partial supervision. The model to be learned, or refined, contains a number of hidden concepts, for which no explicit input or output is provided in the training data. In empirical learning tasks, no initial model is provided and the classification model is learned from scratch. The complexity of the model is constrained by specifying the language in which it is described, e.g. decision tree, conjunctive normal form (CNF), multi-layer perceptron (MLP), etc. In knowledge refinement, the model is provided and needs to be changed. In most cases, the training data contain information for only a subset of the variables in the original model.

In symbolic machine learning there is no explicit representation of the hidden concepts and therefore the distinction between full and partial supervision is not clear. The analogy to neural network learning is more straightforward. The parallel of full supervision is learning a simple perceptron classifier, i.e., a linear discrimination model. Partial supervision resembles the task of learning a multi-layer perceptron (MLP), where no training information for the values of the hidden nodes is provided. Feedback allocation in MLPs is performed by the error back-propagation algorithm.

The learning task in MLPs differs in many respects from the problem examined in this chapter. The goal of error back-propagation is to learn the relation strength between concepts and their subconcepts. Thus, if each node in an MLP is used to represent an event, the weights on the links show the dependence of each event on its subevents. In a TCN, this dependence is provided by the initial model and it is considered to be fixed, i.e., non-refinable. Instead, the aim of refinement in a TCN is the correct setting of the temporal parameters in the model. The representation of temporal parameters in a distributed system like an MLP is an important problem, as explained in chapter 2. A further difference between MLP learning and the TCN refinement is the use of non-differentiable functions, i.e., the logic operators, in the nodes of a TCN. Differentiable activation functions are necessary for the back-propagation algorithm.

In knowledge refinement, the hidden concepts are explicitly specified in the initial model

and therefore the issue of feedback allocation becomes important. In some of these systems, e.g. Expert Networks [49], the adopted approach to feedback allocation is similar to that in MLP learning and the same arguments mentioned above for MLPs apply. The most relevant approaches to the problem examined here are those grouped under the category knowledge base refinement in chapter 2. These are methods which evaluate the performance of the system on the training set and use heuristics to determine which parts of the knowledge base need to be changed. The method presented in this chapter belongs in this category, but it is designed to suit the structure of the TCN and the temporal nature of the event recognition task. Its most important difference with the standard knowledge refinement methods is that the training data are not provided as a set of separated examples, each associated with the desired classification. Instead, the data appears in two parallel streams of event occurrences, one for the input and one for the output nodes in the TCN. The refinement algorithm needs to construct the positive and negative examples, establishing the mapping between event occurrences in the two streams.

6.3 Two-stage refinement

As was seen in the example of section 6.1, the main problem in refining a TCN model under partial supervision is to decide which event occurrences *should be* recognised, using the limited training feedback. This process is performed by the feedback allocation algorithm. However, in order to allocate the feedback, the occurrences of intermediate, i.e., non-output, events which *could be* recognised should be available.

For this reason, the refinement of a TCN under partial supervision consists of two stages: forward belief initialisation, or *classification*, and backward *feedback allocation*. In the first stage all subevent occurrences that can possibly be recognised in the input data are recognised and classified according to the original model. For this purpose an extended version of the event recognition algorithm is used, in which:

- all event sequences that satisfy the structural requirements of an event definition cause the recognition of the event, irrespective of whether the temporal constraints are satisfied,
- the temporal constraints are used in the calculation of an initial *recognition belief* for the event occurrence, i.e., a confidence value on the recognition of the occurrence,
- all intermediate event occurrences, irrespective of their recognition belief can be used

to recognise higher-order ones.

The sequence of the four input events in the example of section 6.1, would cause the recognition of the following event occurrences:

$$\begin{array}{lllll} A(0,2) & & & & \\ B(3,4) & \rightsquigarrow_{-0.11} & W(0,4) & & \\ C(5,7) & & & & \\ D(8,10) & \rightsquigarrow_{0.11} & X(5,10) & \rightsquigarrow_{-0.11} & Y(0,10), \end{array}$$

where occurrences on the RHS of $\rightsquigarrow_b$ are recognised with an initial *recognition belief* $b \in [-1..1]$. If $b > 0$, the initial classification of the example is positive, otherwise it is negative. The value of b corresponds to the confidence in this classification. Section 6.6.2 explains how this confidence is calculated.

In the feedback allocation stage, the feedback for the output event occurrences is used to:

- refine the parameters in the definition of the output events,
- allocate training feedback to the supporting subevent occurrences.

This process is carried out recursively for intermediate events, back to the input nodes of the TCN. Thus, if $Y(0,10)$ ought to be recognised in the above example, it receives an explicit feedback of 1, meaning that it is a positive example of Y. This information is used to label the example as positive and use it as such in the refinement of the temporal parameters in the definition of Y. Furthermore, it is used to allocate feedback to the supporting occurrences of W and X. In this case, both $W(0,4)$ and $X(5,10)$ should receive positive feedback. Algorithm 6.1 provides an overview of the two-stage refinement algorithm under partial supervision (RAPS). The following sections describe the individual algorithms in detail, pseudocode descriptions of which are given in appendix E.

6.4 Forward propagation of recognition beliefs

6.4.1 Causality in the relative event support trees

In order to allocate training feedback to the intermediate events in the TCN, it is necessary to maintain information about the support for each event occurrence. For this purpose, the causal dependencies between the event occurrences are stored in the RESTs of the corresponding events. These dependencies can be represented as causal graphs the nodes of which correspond

Input: the model; input event occurrence; training feedback[a]; mode of operation; the relative event support trees (RESTs); the node histories.
Output: the refined model.
RAPS(model,event,feedback,mode):
Mode of operation A: classification

a. Recognise all event occurrences that can be recognised, ignoring the temporal constraints.
b. Classify each event occurrence as positive or negative, according to whether it satisfies the temporal parameters.
c. Assign a recognition belief to each event occurrence.
d. Update the RESTs and the node histories for each of the recognised events.

Mode of operation B: refinement

1. `FOR EACH` output event occurrence `DO`:
   a. Check the training data for this event occurrence. If it should be recognised assign feedback 1 to the corresponding example, otherwise assign -1.
   b. Refine the temporal constraints in the definition of the event, using the REST for the event and the training feedback for each example.
   c. Use the feedback and the model-based recognition beliefs to calculate the feedback for each of the supporting subevent occurrences.[b]
2. `FOR EACH` intermediate event occurrence, that has received feedback `DO`:[c]
   a. Refine the temporal constraints for the event definition.
   b. Calculate the feedback for each of the supporting subevent occurrences.

[a]The training feedback includes all output events that should be recognised up to the current point in time.
[b]Input event occurrences are not assigned any feedback.
[c]Intermediate events are processed, according to the inverse partial support order, so that all events which are supported by another are examined before it and may assign feedback to it.

Algorithm 6.1: Overview of the two-stage refinement process under partial supervision.

to the examples of events, i.e., event occurrences, and the links to the supporting relations between them. Figure 6.2 presents the causal graph for the example of section 6.1. The label of each node in the graph contains:

a. the signature of the event, e.g Y,
b. the unique example label, which is assigned to each non-input event occurrence when it is recognised, e.g. **3**, and
c. the duration of the recognised event occurrence, e.g. 10.

The duration of the event occurrence is needed for distinguishing between alternative begin points for the same example. Due to the way in which event occurrences are recognised in a TCN, each example label is associated uniquely with the terminal subevent occurrence and therefore the end point of the recognised event. However, it may correspond to more than one

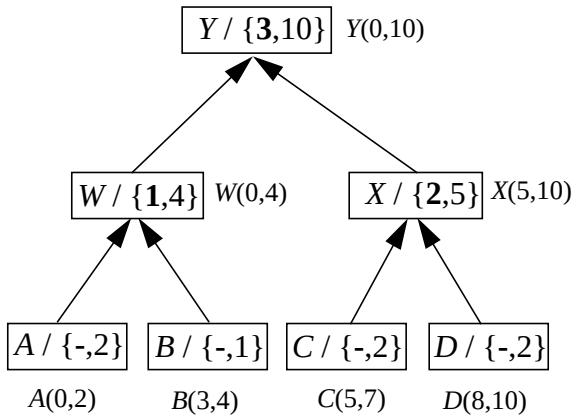


Figure 6.2: Simple causal support graph. Node format: *signature*/{**example**,duration}. In order to distinguish between example labels and temporal parameters, the former are shown in boldface.

begin point and, even for the same begin points, to more than one subevent sequence. In the example used above, if another occurrence of A, $A(2,3)$, was recognised, then examples **1** of W and **3** of Y would have two possible begin points: 0 and 2. If in addition a second occurrence of C was recognised, $C(7,9)$, then example **2** of X would also have two possible begin points: 5 and 7. The output event Y could then be recognised with four different combinations of its subevents, which pairwise have the same begin and end points:

$$W(0,4), X(5,10) \quad \& \quad W(0,4), X(7,10),$$
$$W(2,4), X(5,10) \quad \& \quad W(2,4), X(7,10).$$

In order to allocate the feedback correctly, intermediate event occurrences of different duration which are assigned the same example label, should be identifiable, because they are treated differently by the supported event. Thus, if $Y(0,10)$ receives positive feedback and $Y(2,10)$ negative, the first of the above pairs of W and X event occurrences should be allocated positive and the second negative feedback.

In order to decide on the feedback to be allocated to subevent occurrences, the causal support information of the graph in Fig. 6.2 is stored in the REST. Each example in the example list at each REST node holds now information about the supporting subevent occurrence. Under full supervision, the examples of events are separated into positive and negative and stored into different example lists in the REST nodes. Under partial supervision, the correct classification

of each example is not known. Therefore, a single list of examples is maintained at each node. Each element of this list is a tuple, containing:

a. the example label,

b. an example weight, which is assigned to each example in the feedback allocation stage and indicates whether the example is positive or negative according to the feedback data,

c. the subevent example label.

Figure 6.3 shows the RESTs for the three non-input events in the example above. The example list in the leftmost node of REST(Y), at the level of subevent W, contains the example $(\mathbf{3}, -1.0, \mathbf{1})$, which is part of example **3** for Y, has a weight of -1.0, meaning that it is definitely negative, and is supported by example **1** of W. Note that the duration of the supporting subevent example is given by the duration of the node, in this case 2. Looking at REST(W) it can be seen that the example with duration 2 corresponds to the left branch of the REST, consisting of an A occurrence of duration 1 and a B of the same duration. The distance between the two event occurrences is 0. The examples in REST(W) and REST(X), do not specify the supporting example label, because their subevents are input events.

The calculation of the example weights is explained in section 6.6.3. The weights are needed for the calculation of purity in the refinement of the temporal parameters. The purity function is very similar to that used under full supervision in chapter 5 and is described in section 6.6.

6.4.2 Model-based recognition belief

In the forward classification stage, *all* event occurrences which can be recognised are generated, ignoring the temporal constraints of the model. If the temporal parameters were used in this stage, the recognition of an event occurrence could be prevented by incorrect parameter values. Occurrences which are not generated cannot be allocated feedback and therefore the erroneous parameters would not be corrected. The initial temporal constraints are used to propose a classification for the event occurrences that are recognised and calculate their *model-based* recognition belief. This belief value provides an initial confidence on the classification of the event occurrence. It measures the proximity of an example to the original model parameters, which gives an indication of the amount of change needed in order to make a positive example

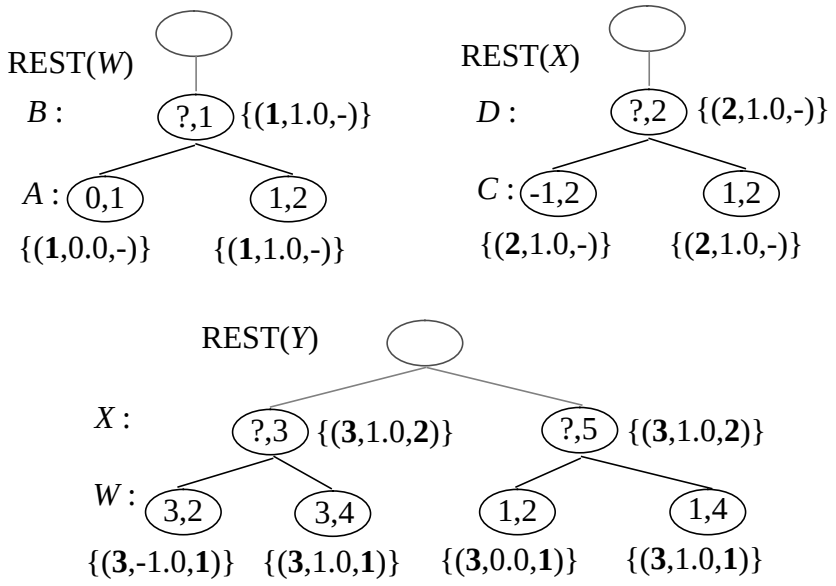


Figure 6.3: RESTs containing causal information. Each node contains a list of examples of the format: (**example**,weight,**subevent example**).

negative and vice versa. The closer the example lies to the boundaries of the model parameters, the easier it is to change its classification by modifying the parameters and therefore the lower the confidence in the initial classification should be. Model-based recognition beliefs are intended to be used as a measure of the quality of the support provided by an event occurrence to higher-order ones and are included in the calculation of the recognition belief for the latter occurrences. Recognition beliefs are not directly interpretable into example weights, which are calculated on the basis of training feedback.

As a means of illustrating the calculation of the model-based recognition belief, assume the following initial model parameters for the three events in the working example:

$$\texttt{duration}(W, A, [1..1]), \quad \texttt{distance}(W, A, B, [-1..2]), \quad \texttt{duration}(W, B, [2..4]),$$
$$\texttt{duration}(X, C, [1..3]), \quad \texttt{distance}(X, C, D, [0..1]), \quad \texttt{duration}(X, D, [1..4]),$$
$$\texttt{duration}(Y, W, [3..6]), \quad \texttt{distance}(Y, W, X, [1..5]), \quad \texttt{duration}(Y, X, [4..7]).$$

The recognition belief for an example of the three non-input events, i.e., W, X and Y, is calculated from two pieces of information:

- The proximity of the example to the original model.

- The recognition belief of the supporting subevent occurrences.

Consider, for instance, example $W/\{\mathbf{1}, 4\}$, i.e., example **1** of W, with duration 4. This example is stored in the right branch of REST(W), in Fig. 6.3, which contains the nodes $B : (?, 1)$ and $A : (1, 2)$. The supporting subevent occurrences for A and B have a recognition belief of 1, since they are input ones, i.e., they have definitely occurred. In this case, the recognition belief for the example depends only on its proximity to the model. The calculation of this proximity can be further subdivided into the calculation of the proximity for each parameter value in the REST branch. The function which performs this calculation is named *cf-prox* and takes the following generic form:[1]

$$\textit{cf-prox}(v, v_1, v_2) : (\mathbb{Z} \bigcup \{?\})^3 \rightarrow [-1..1], \tag{6.1}$$

where v is either the distance or the duration value of the REST node and $[v_1..v_2]$ the corresponding parameter range. Thus, in order to calculate the recognition belief for $W/\{\mathbf{1}, 4\}$, the result of the following three function evaluations needs to be combined:

$$\textit{cf-prox}(2, 1, 1) \ \& \ \textit{cf-prox}(1, -1, 2) \ \& \ \textit{cf-prox}(1, 2, 4),$$

using the parameter values in the REST nodes and the parameter ranges of the model. Due to their use in calculating the recognition belief for the example, each *cf-prox* value is called *partial* recognition belief and provides:

- the classification for one parameter value in the example, i.e., positive if it is outside the parameter range and negative otherwise, and
- the confidence in this classification, i.e., an absolute belief value, depending on how close the parameter value lies to the boundary of the parameter range.

It takes lower negative values the further the parameter value is outside the range and takes higher positive values the closer the parameter value is to the middle of the ranges. In the above example, the distance between the two event occurrences is within the required range, but their durations are not. As a result, the proposed classification is negative for both durations and positive for the distance value, i.e., *cf-prox*$(2, 1, 1) < 0$, *cf-prox*$(1, 2, 4) < 0$, *cf-prox*$(1, -1, 2) > 0$.

The set of partial recognition beliefs for the parameter values in an event example give an indication of the proximity of the example to the model parameters. This set is combined with

[1] *cf-prox*: confidence based on proximity.

the set of recognition beliefs for the supporting subevent occurrences, to provide the recognition belief for the example:[2]

$$\textit{conj-rb}(P, Q) : [-1..1]^{(2n-1)} \times [-1..1]^n \rightarrow [-1..1], \tag{6.2}$$

where P is the set of partial recognition beliefs and Q the recognition beliefs of the n supporting events. In the working example:

$$\begin{aligned} P &= \{\textit{cf-prox}(2,1,1), \textit{cf-prox}(1,-1,2), \textit{cf-prox}(1,2,4)\}, \\ Q &= \{1,1\}. \end{aligned}$$

The recognition beliefs, set Q, for the input event occurrences are both 1. In order for the example to be positive, all partial beliefs and the recognition beliefs for the supporting subevent occurrences, must be positive. This is not the case in this example, because the duration constraints are not satisfied and therefore the overall result is negative.

Finally, if there was more than one branch of REST(W) with a total duration of 4 for the event example **1**, a further function would be used:[3]

$$\textit{disj-rb}(R) : [-1..1]^n \rightarrow [-1..1], \tag{6.3}$$

where R is now the set containing the result of *conj-rb* for different REST branches. For instance, example $Y/\{\mathbf{3}, 10\}$ in REST(Y) corresponds to two alternative branches: $\{X : (?, 3)$, $W : (3, 4)\}$ and $\{X : (?, 5)$, $W : (1, 4)\}$. The function *conj-rb* provides a recognition belief for each sequence and the results are combined, using *disj-rb*.

The result of this process is the recognition belief of an example and is used in the calculation of the recognition belief for the examples of higher-order events. Figure 6.4 summarises the calculation of the model-based recognition belief for $W/\{\mathbf{1}, 4\}$. First the partial recognition beliefs for the three parameter values are calculated, using *cf-prox*. Then *conj-rb* is used to combine the partial recognition beliefs and the recognition beliefs of the supporting subevent occurrences. Finally, *disj-rb* provides the recognition belief for the example, combining the belief for alternative branches, in this case just one. The details of the three functions are presented in section 6.6.2. The resulting recognition belief for $W/\{\mathbf{1}, 4\}$ is used to calculate the recognition belief for $Y/\{\mathbf{3}, 10\}$.

[2] *conj-rb*: conjunctive combination of recognition beliefs.
[3] *disj-rb*: disjunctive combination of recognition beliefs.

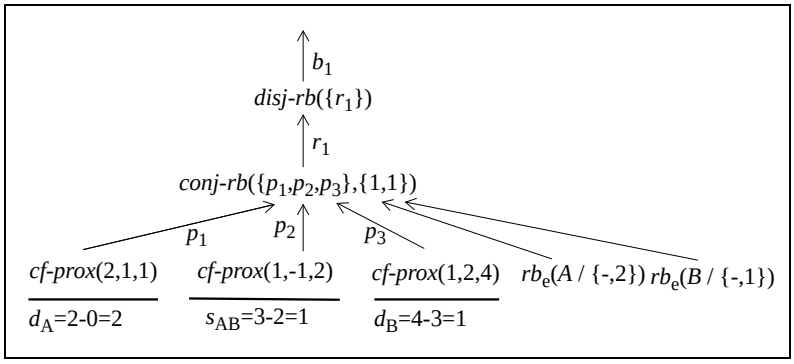


Figure 6.4: Calculation of the model-based recognition belief for $W/\{\mathbf{1}, 4\}$. The rb_e function provides the recognition belief for an event example.

6.5 Backward allocation of feedback

When an output event is recognised, training feedback for the event occurrence is acquired and the feedback allocation process can take place. This process is the same for all non-input events and consists of three stages:

1. Aggregating the received feedback for each event example.
2. Updating the example weights in a REST (*Intra-REST allocation*).
3. Allocating feedback to the supporting subevent occurrences (*Inter-REST feedback*).

6.5.1 Feedback aggregation

In section 6.4.2 the calculation of an initial model-based recognition belief for an event occurrence was examined. In the feedback allocation stage, the occurrence is assigned feedback from the data, if it is an output event, or from the event occurrences that it supports, if it is an intermediate event. Input events do not receive feedback. Feedback values are in the same range as recognition beliefs, i.e., $[-1..1]$, and are aggregated to provide a *data-based* estimate for the belief in the recognition of an event.

The feedback received from different supported event occurrences is aggregated by the function:[4]

$$\textit{rb-combine}(F) : [-1..1]^n \rightarrow [-1..1], \tag{6.4}$$

[4] *rb-combine*: recognition belief combination.

where F is the set of all the feedback values assigned to an event occurrence and n, the number of event occurrences that have allocated feedback to it. For output event occurrences, a single feedback value is received ($n = 1$), which is either -1, if the event should not be recognised, or 1 if it should be. For intermediate event occurrences $n \geq 1$ and some of the feedback that is received may be contradictory. In other words, one supported event occurrence may assign positive and another negative feedback to their supporter. In the working example, there is only one output event, Y, allocating feedback to W and X. Assume that the two occurrences of Y receive different feedback: $Y(0, 10)$ receives $+1$ and $Y(2, 10)$, -1.

6.5.2 Intra-REST allocation

Once the data-based recognition belief for an event example, e.g. $Y/\{\mathbf{3}, 10\}$, which corresponds to occurrence $Y(0, 10)$, has been calculated, its classification information, represented by the weights in REST(Y), needs to be set accordingly. Similar to recognition beliefs, positive weights correspond to positive and negative to negative examples. Moreover, the higher/lower the data-based recognition belief for a positive/negative example, the higher/lower the weight that this example is awarded.

The data-based beliefs for the two Y occurrences are $f_{31} = 1$ for $Y(0, 10)$ and $f_{32} = -1$ for $Y(2, 10)$.[5] Each of these occurrences corresponds to two branches of the REST(Y), i.e., two different subevent sequences. For example, $Y(0, 10)$ corresponds to the branches $\{X : (?, 5), W : (1, 4)\}$ and $\{X : (?, 3), W : (3, 4)\}$. The first requirement is to allocate feedback to each separate branch. In other words, knowing that $Y(0, 10)$ is positive, $f_{31} = 1$, the feedback for $\{X : (?, 5), W : (1, 4)\}$, f_{311}, and $\{X : (?, 3), W : (3, 4)\}$, f_{312} needs to be calculated.[6] One of the two sequences is required to be positive and should receive strong positive feedback. The recognition of the second sequence of subevent occurrences is not needed and therefore weaker feedback can be awarded.

The allocation of feedback to alternative sequences for the same example is done on the basis of their model-based belief. The rationale is that a sequence which has a high model-based recognition belief is more appropriate for receiving high positive feedback and less likely

[5]The index for the beliefs contains the example number, which is **3** in this case and a number to distinguish between the event sequences.

[6]The additional index is needed for distinguishing between different sequences for the same example and with the same duration.

to be negative. Feedback allocation is performed by the function:[7]

$$\textit{inv-disj-rb}(R, f) : [-1..1]^{(n+1)} \rightarrow [-1..1]^n, \tag{6.5}$$

where R is the set of model-based recognition beliefs for the n alternative subevent sequences and f is the data-based recognition belief of the event occurrence. The recognition beliefs in R are the output of *conj-rb*, which is an intermediate product in the classification stage, i.e., before the combination of recognition beliefs for separate branches of the REST, by *disj-rb*.

In the above example from Fig. 6.3 there are two alternative subevent sequences for *each* occurrence of Y, i.e., $n = 2$, and assume that the following beliefs are calculated:

$$\begin{aligned}
f_{311} &= 1.0, \text{ for the nodes } \{X : (?, 5), W : (1, 4)\}, \\
f_{312} &= 1.0, \text{ for the nodes } \{X : (?, 3), W : (3, 4)\}, \\
f_{321} &= 0.0, \text{ for the nodes } \{X : (?, 5), W : (1, 2)\}, \\
f_{322} &= -1.0, \text{ for the nodes } \{X : (?, 3), W : (3, 2)\}.
\end{aligned}$$

The details of how these belief values are calculated are given in section 6.6. The calculated beliefs can effectively be used as the weights of the corresponding examples. For instance, nodes $X : (?, 5)$ and $W : (1, 4)$, in Fig. 6.3, have a weight of 1.0, which is provided by f_{311} above. Similarly, $W : (3, 2)$ has a negative weight, because f_{322} is negative.

A final complication is that more than one of the above sequences may share the same REST node. This is true for the nodes of the terminal subevent, X, e.g. $X : (?, 5)$ is used in two of the four branches:

$$\{X : (?, 5), W : (1, 4)\} \ \& \ \{X : (?, 5), W : (1, 2)\}.$$

Since each subevent sequence may have a different recognition belief, a node-specific calculation of example weights is also needed. Thus, the example weight of node $X : (?, 5)$ should combine the feedback of the two subevent sequences above, i.e., $f_{311} = 1.0$ and $f_{321} = 0.0$. The node-specific weight combination is performed by the following function:[8]

$$\textit{comb-weight}(F) : [-1..1]^n \rightarrow [-1..1], \tag{6.6}$$

[7] *inv-disj-rb*: inverse of *disj-rb*.
[8] *comb-weight*: combined weight.

where F is the set of feedback values for the n branches which share the node. Applying this function to the nodes of REST(Y) results in the weights that appear in the REST nodes in Fig. 6.3.

6.5.3 Inter-REST feedback

The final stage of the refinement process involves the allocation of feedback from an event occurrence to its supporters. As mentioned in the previous section, feedback values are in the range $[-1..1]$ and are interpreted as data-based recognition beliefs. Positive feedback is assigned to event occurrences which should be recognised and negative to the ones that should not.

In the simplest case, an output event occurrence, e.g. $Y(0, 10)$, assigns feedback to its intermediate supporters, e.g. $W(0, 4)$. Positive output examples should try to increase the recognition belief of their supporters, so that the latter can be recognised. This can be achieved by assigning positive feedback to them. Thus, since $f_{31} = 1$ for $Y(0, 10)$, the feedback to $W(0, 4)$ should be positive, to ensure that $W(0, 4)$ is recognised. Negative examples should discourage the recognition of their supporters, assigning negative feedback to them. Note, however, that the recognition of a negative example can be avoided even when all of its supporters have a positive recognition belief. This can be achieved by setting the temporal parameters of the event appropriately. Thus, if $W(2, 4)$ is not excluded by the temporal parameters for W, the recognition of $Y(2, 10)$ can be avoided by setting the temporal parameters of Y appropriately. The opposite is true for positive examples, where the recognition of an event requires *all* of its supporters to be positive and *all* the temporal constraints to be satisfied. This fact makes the maintenance of positive feedback particularly important and one of the major factors in the choice of a feedback allocation function in section 6.6.

The feedback value for intermediate event occurrences is not $+1$ or -1, but any value in the range $[-1..1]$. For instance the occurrence $W(2, 4)$ receives feedback from the occurrences of event Y. If W was supported by other intermediate subevents, i.e., if A and B were not input events, it would need to allocate feedback to them. This feedback should be proportionate to the data-based recognition belief for $W(2, 4)$, i.e., the belief in the recognition of $W(2, 4)$, after the feedback from Y has been received. Thus, the allocated feedback contains more information than just the classification of the supporter, i.e., whether a supporter of $W(2, 4)$

is positive or negative. It represents the belief in the recognition of the subevent occurrence, given the recognition belief of the supported one.

The calculation of example weights, as explained in section 6.5.2 above, combines the data-based recognition belief for the supported event occurrence and the model-based belief for subevent sequences. For this reason, the example weights are a good estimate of the feedback to be allocated to the supporting subevent occurrences. For instance, the feedback allocated to $W/\{\mathbf{1},2\}$, i.e., the occurrence $W(2,4)$, is a combination of the weight for examples supported by $W/\{\mathbf{1},2\}$, i.e., examples in the nodes $W:(3,2)$ and $W:(1,2)$ in REST(Y), Fig. 6.3. In order to combine weights from different nodes and provide a feedback value for the supporting subevent example, the *comb-weight* function (6.6) is used again. The feedback for the four subevent occurrences supporting Y, in Fig. 6.3, is as follows:

$$\begin{aligned} f(W/\{\mathbf{1},4\}) &= 1.0, \\ f(W/\{\mathbf{1},2\}) &= 0.0, \\ f(X/\{\mathbf{2},5\}) &= 1.0, \\ f(X/\{\mathbf{2},3\}) &= 1.0. \end{aligned}$$

The details of how these values are calculated are provided in section 6.6.3.

Once the feedback to these events has been allocated, the first two stages of the backward allocation process are repeated for each of them, resulting in the weights of Fig. 6.3. These can then be used in the refinement of the temporal constraints for events W and X. Figure 6.5 summarises the use of the feedback allocation functions for the working example. The feedback received for each of the two output event occurrences, i.e., $+1$ for $Y/\{\mathbf{3},10\}$ and -1 for $Y/\{\mathbf{3},8\}$, is not altered by *rb-combine*. This is always the case for output event occurrences, which receive a single number as feedback. Then *inv-disj-rb* combines the feedback value with the model-based belief of each REST branch, e.g. $\{X:(?,5),W:(1,4)\}$, to calculate the data-based belief for the corresponding subevent sequence. The result of this function is a set of values, one for each branch, which are interpreted as example weights. If more than one branches with different weights share the same node, *comb-weight* calculates the node-specific weight for the example. Finally, in order to calculate the feedback for each supporting subevent occurrence, the weights of all the examples that the occurrence supports are combined again by *comb-weight*.

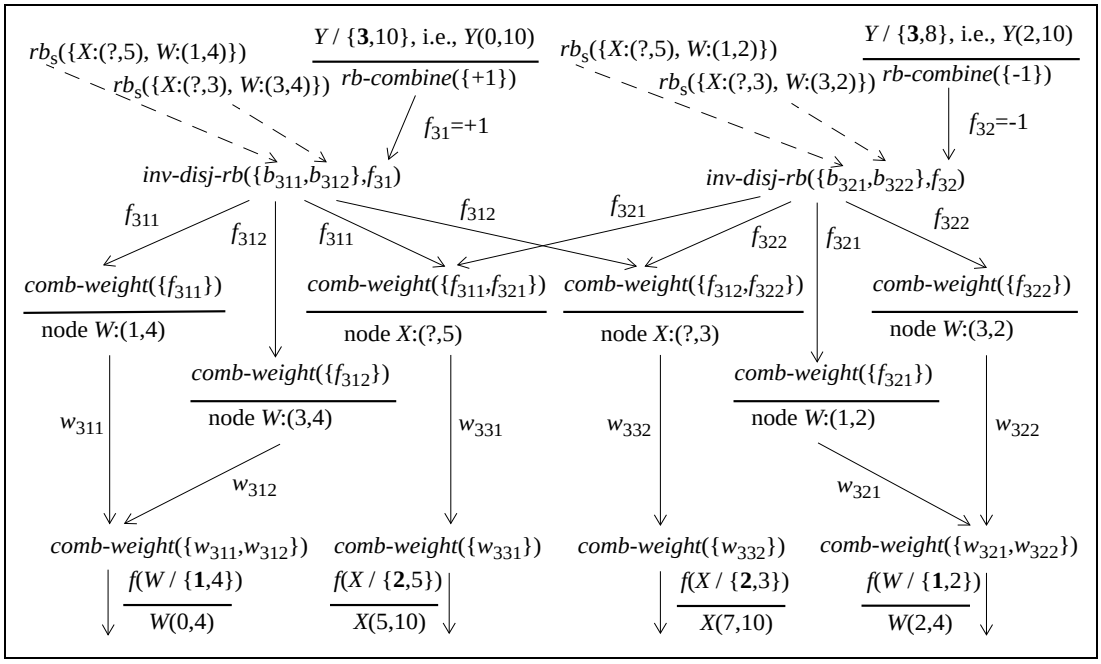


Figure 6.5: Allocation of the feedback received for $Y/\{\mathbf{3},8\}$ and $Y/\{\mathbf{3},10\}$. The function rb_s calculates the model-based belief for one branch of the REST, i.e., one of the alternative sequences corresponding to an example. In that respect it differs from rb_e used in Fig. 6.4.

6.6 Belief processing functions

In the previous section several functions were introduced to perform the initialisation and updating of recognition beliefs. The choice of these functions is as important for the success of the refinement method as the choice of the search heuristics, i.e., proximity and purity, used under full supervision. In this section the criteria for choosing the belief processing functions and the properties of the ones that have been chosen are examined. The functions are separated into three groups, depending on the purpose they serve:

1. Forward belief initialisation. These are the functions that calculate the initial model-based belief for the recognised events.
2. Backward feedback allocation. Functions that allocate the feedback received for the output events, back to the intermediate ones, and recursively further back.
3. Parameter refinement. These are the search heuristics, used for the refinement of the model parameters.

6.6.1 Criteria

In addition to the function-specific criteria, which are discussed in the following sections, there are a number of general issues, which affect the choice of the functions:

Positive against negative examples. As mentioned in section 6.5.3, the recognition of positive examples sets different constraints to the feedback allocation process than that of negative ones. In order for an output event occurrence to be recognised, *all* of its supporters must have been recognised, i.e., have positive recognition beliefs, and *all* of its temporal parameters must be satisfied by the supporting subevent sequence. The same holds for each of their subevents and so on back to the input nodes of the network. Thus, there is a large set of conditions that need to be satisfied to achieve a single positive classification. In contrast, recognition of negatives can be avoided in a number of different ways, e.g. by setting a single temporal parameter appropriately.

The first effect of this difference between the two example types is that the number of examples initially classified as negative is larger than the positive ones. This will be especially true at the higher-order nodes of the network, due to the cumulative effect of parameter errors. This phenomenon can even be observed in the simple working example used in this chapter. Event Y cannot be recognised, because of an error in the temporal parameters of W, which leads to negative recognition beliefs for all W event occurrences. Under the assumption that the original parameters are close to the desired ones and therefore only minor modifications to the model are needed, the value of these negative recognition beliefs should be close to zero. However, this depends very much on the choice of the belief initialisation and propagation functions.

The difference between positive and negative examples affects also the choice of feedback allocation functions. Since the satisfaction of positives is harder than the exclusion of negatives, positive feedback must be carefully maintained in the backward allocation stage.

Diminishing feedback. A problem related to the special importance of positive examples is that of diminishing feedback in the backward allocation stage. The feedback assigned by an event occurrence to its supporters is proportional to the feedback it has received. In a high-order network, this can result in diminishing feedback values, which become very low for event

occurrences at the lower-order nodes. This effect can be justified by the increasing uncertainty as causality of recognition is traced back from the output to the input nodes. An example of this uncertainty can be seen in the working example, where either of the two X events, $X(5, 10)$ and $X(7, 10)$, can be used to recognise the required $Y(0, 10)$. It is not clear which of the two X occurrences should be associated with the recognition of $Y(0, 10)$ and receive positive feedback.

The decrease in the feedback values is reflected in the weights of examples, which are used in refinement. This can be particularly harmful for positive examples, which need to have considerably larger weights than negative ones, in order to guide the search to good solutions. Thus, it is important to choose the feedback allocation functions in such a way as to minimise the effect of diminishing feedback.

Weighted intra-REST allocation. Another effect of the uncertainty in the backward trace of causality, is the distinction between alternative solutions. In the example that was given above, there are two X event occurrences that can be used to recognise Y. As a result, they are both awarded positive feedback. In reality, however, only one of them is likely to be positive. Thus, a way of discriminating between them is needed. To some extent this can be achieved, by taking into account the original model-based recognition belief of the two X events and a measure of how well they satisfy the temporal constraints of Y, given by the *cf-prox* function. A combination of the two measures is given by the *conj-rb* function. The higher the model-based recognition belief of the event occurrence, the higher the positive feedback that it should receive.

Data-based feedback and model-based recognition beliefs. Under full supervision, training feedback was purely data-based and was only combined with model-based information by the explicit combination of the proximity and purity heuristics. The advantage of this is that the relative importance of the two sources of information can be simply weighted, according to their importance in the refinement of the model. As seen above, however, the feedback allocation process uses model-based information. The extent and the manner in which this is done is important for the choice of the feedback allocation and refinement functions.

6.6.2 Forward belief initialisation

In the forward classification stage of the refinement process, the model-based initialisation of recognition beliefs takes place. In the centre of this process there is the *cf-prox* function, which calculates a belief in the proposed classification, based on the proximity of an example to the model parameters. Figure 6.6 shows the two sequences:

$$k_1 = A(0,2), B(3,4) \ \& \ k_2 = A(2,3), B(3,4),$$

for the example **1** of event W, in the three-dimensional parameter space. Both sequences fail to satisfy the constraint on the duration of B, but sequence k_1 also fails to satisfy the duration of A.

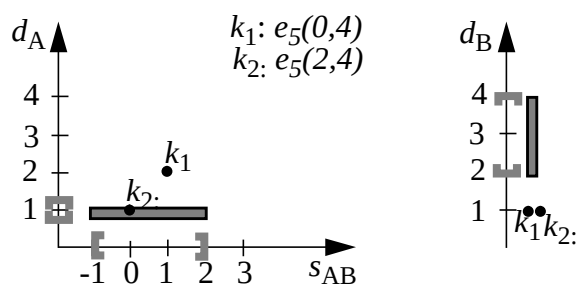


Figure 6.6: An example of two event sequences represented in the same space as the parameters of the model. The shaded area corresponds to the region covered by the model and the bracketed part of each axis to the projection of the covered area. The nodes of the REST(W) are represented as dots.

The *cf-prox* function calculates a belief for each parameter value, named the partial recognition belief. The partial recognition belief is positive, if the parameter value is within the given range and negative outside it. Positive values are higher when they are close to the middle of the range and negative ones are smaller when they are far outside the boundaries. The parameter ranges are inclusive and therefore examples on the boundaries should be assigned a positive belief. Moving one time unit outside the ranges, they become negative. Thus the belief is always different from zero. Regarding the minimum belief value, it can either approximate -1 as the proximity to the boundaries decreases, or it can be set to -1 at the window limits. The maximum belief value is at the middle of each range and is set to 1, in order to reward good positive examples.

The function that has been chosen for *cf-prox* is inspired by the Lennard-Jones potential, used to calculate attraction/repulsion forces between molecules. Ignoring initially the sign of the belief values, (6.7) provides a decreasing positive value when moving from the middle of the range, $(\frac{v_1+v_2}{2})$, to the perimeter, and increases again when moving away from the perimeter.

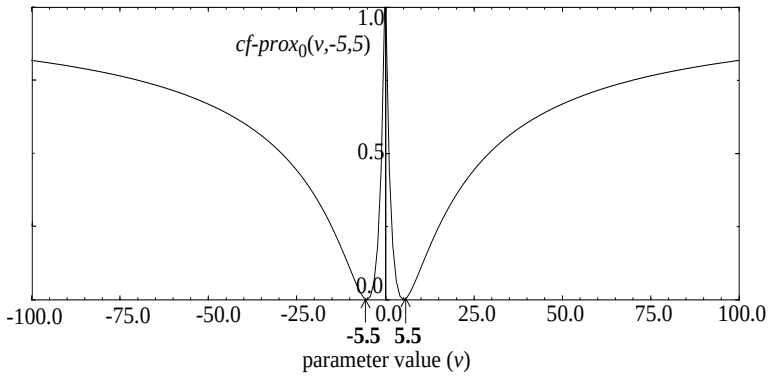
$$\textit{cf-prox}_0(v, v_1, v_2) = 1 - \frac{4 \times (v_2 - v_1 + 1) \times |2v - v_1 - v_2|}{(v_2 - v_1 + 1 + |2v - v_1 - v_2|)^2}. \tag{6.7}$$

The *cf-prox*$_0$ function provides a measure of confidence in the classification of an event sequence with the particular parameter value v. It indicates how safe the proposed classification – whether positive or negative – is. Figure 6.7(a) illustrates the behaviour of the function for $v_1 = -5$, $v_2 = 5$. The *cf-prox* function can be derived from *cf-prox*$_0$ by multiplying its output with -1 for negative examples. Figure 6.7(b) shows the effect of this for the function in Fig. 6.7(a). This function satisfies all of the requirements that were set for *cf-prox* and also provides clear discrimination between "good" positive and "bad" negative examples, due to its non-linearity.

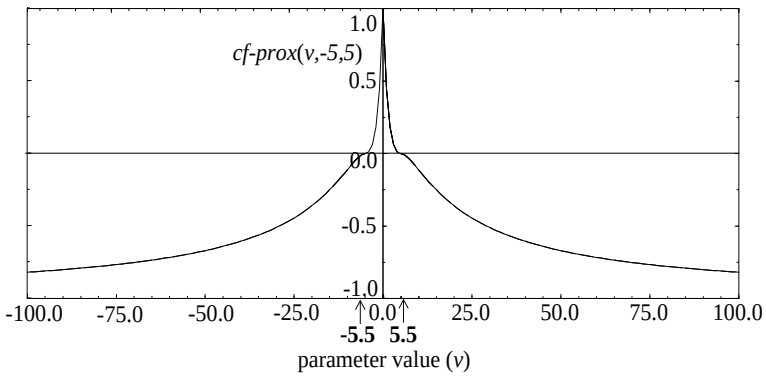
A linear approximation of the above function is a combination of two linear segments with negative slope. In order for the values of this function to remain above -1, upper and lower bounds to v need to be introduced. The natural choice is the parameter window. Thus, assuming that the window for the function presented in Fig. 6.7(b) is $[-100..100]$, Fig. 6.7(c) presents the linear alternative. Typically, the slope for the negative segment will be less steep than the positive one, since the window of interesting values for a parameter is expected to be considerably larger than the original range. The result of this is increased uncertainty in the assignment of negative recognition beliefs. This can only be avoided by choosing a narrow window.

Another alternative is to view the *cf-prox* function as a probability density function, indicating the probability of an example being positive. For example, one could assume that positive examples are normally distributed, with a mean at the middle of the parameter range and the variance equal to the original range. A practical problem with this approach is that positive examples are not assigned high enough probabilities to be clearly distinguished from negative ones.[9]

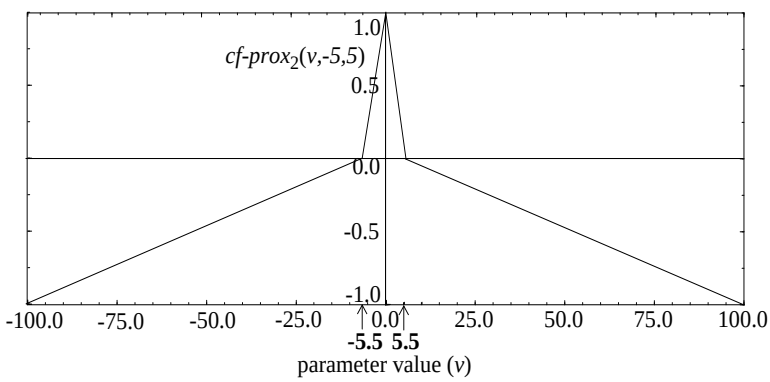
[9]This alternative has not been examined further, due to lack of time.



(a) Ignoring the sign of the belief value. (*cf-prox*$_0$)



(b) Multiplying by -1 for negatives.



(c) A linear alternative.

Figure 6.7: The *cf-prox* function for a range of size 10.

The partial recognition beliefs, calculated by *cf-prox* for the parameter values of an example, are combined with the recognition beliefs of the subevent occurrences to give the model-based recognition belief for the example. Since the recognition of the sequence requires both positive supporters and the satisfaction of the model parameters, the combined recognition belief, ought to be no higher than the lowest of the individual beliefs. The combination function used here takes the minimum of all the belief values, similar to the conjunctive combination of confidence factors [94]. This is preferred to probabilistic combination, i.e., multiplication of the two beliefs, which can lead to a pessimistic estimate of the recognition belief. Thus,

$$\textit{conj-rb}(P, Q) = \min(P \bigcup Q), \tag{6.8}$$

where P the partial recognition beliefs and Q the recognition beliefs for the subevent occurrences. The *conj-rb* function provides an initial recognition belief for an event example with a particular duration. A practical problem with the use of the minimum belief is that the recognition belief for positive examples can quickly become very low, in the forward propagation of beliefs. This problem can be solved to a certain extent by the way in which model-based recognition beliefs are used in the feedback allocation stage.

The last function used in the forward classification stage is *disj-rb*, which calculates the combined recognition belief for a set of alternative sequences of the same example, with the same duration. Since the recognition of the particular example requires only one of the alternatives to be recognised, the belief in its recognition has to be at least as high as each of the individual beliefs. Following again the confidence-factor methodology, the maximum is chosen for this function.

$$\textit{disj-rb}(B) = \max B. \tag{6.9}$$

Thus, if there is more than one way of achieving a positive example, the recognition belief is inherited from the most optimistic alternative.

Figure 6.8, reproduces the example of Fig. 6.4, replacing the variables with numbers.

6.6.3 Backward feedback allocation

The feedback allocation process begins with the calculation of a data-based recognition belief for an event occurrence, combining all the feedback received either from the training data or

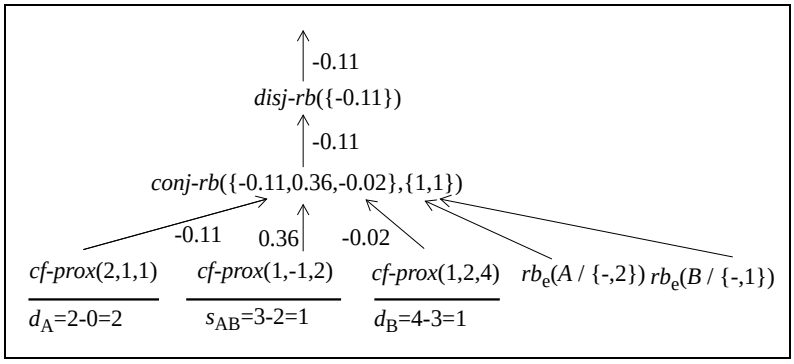


Figure 6.8: Calculated belief values of the model-based recognition belief for $W/\{\mathbf{1}, 4\}$.

from the event occurrences that it supports. The combination of feedback is performed by the *rb-combine* function, which is chosen to provide a bias for positive classification. This bias is achieved by selecting the maximum among the received feedback values:

$$\textit{rb-combine}(F) = \max F, \tag{6.10}$$

The choice of enforcing a positive bias is justified on the basis of the preservation of feedback for positive examples, as explained in section 6.6.1 above.

A problem with the chosen function is that it does not provide adequate discrimination between positive and negative examples, since an example that receives a single positive piece of evidence is considered positive. A pseudo-probabilistic alternative is to treat the feedback as evidence provided by independent sources about the recognition of the event occurrence and use probabilistic summation of the evidence.

$$\textit{rb-combine}(F) = \sum_{i=1}^{n} f_i^+ - \sum_{j=1}^{m} |f_j^-|, \tag{6.11}$$

where $f_i^+ \in F$ is the positive feedback, $f_j^- \in F$ the negative and $\sum$ stands for probabilistic summation. For instance, if the example $W/\{\mathbf{1}, 4\}$ received feedback $f_1 = 1.0$, $f_2 = 0.5$, $f_3 = -0.5$ and $f_4 = -0.3$, the combined feedback would be:

$$\begin{aligned} \textit{rb-combine}(\{1.0, 0.5, -0.5, -0.3\}) &= (1.0 + 0.5 - 1.0 \times 0.5) - (0.5 + 0.3 - 0.5 \times 0.3) \\ &= 1.0 - 0.65 = 0.35 \end{aligned}$$

One potential problem with the use of probabilistic summation is the handling of contradictory evidence. A single piece of negative evidence can result in a considerable reduction in

the recognition belief of an event occurrence, even when all other evidence is positive. This problem can be overcome, by scaling the evidence according to the size of the two subsets of F, F^+ and F^-, corresponding to positive and negative evidence. Furthermore, a user-defined weight ζ for the relative importance of positive and negative evidence can be added:

$$\textit{rb-combine}(F) = \frac{\zeta n}{n+m} \mathop{\circ\!\!\sum}_{i=1}^{n} f_i^+ - \frac{(1-\zeta)m}{n+m} \mathop{\circ\!\!\sum}_{j=1}^{m} |f_j^-|. \tag{6.12}$$

The positively-biased alternative presented in (6.10) was preferred on the grounds of simplicity.

The next stage involves the allocation of feedback to alternative sequences for the same example, with the same duration. For example, the two different sequences, corresponding to the event occurrence $Y(0,10)$:

$$W(0,4), X(7,10) \ \& \ W(0,4), X(5,10)$$

should be allocated different feedback. In fact it is likely that one of the two sequences is a negative example, despite the fact that $Y(0,10)$ should be recognised. The two alternative subevent sequences are represented in REST(Y), Fig. 6.3, by the branches:

$$\{X:(?,3), W:(3,4)\} \ \& \ \{X:(?,5), W:(1,4)\}.$$

As was discussed in section 6.5.2 above, the function *inv-disj-rb* is used for allocating feedback to each individual sequence. This function scales the feedback allocated to $Y(0,10)$, by the model-based beliefs of the individual functions. This is achieved as follows:

$$\textit{inv-disj-rb}(R,f) = \left\{ f_i : f_i = \begin{cases} f, & \text{if } \max R = \min R \text{ or} \\ f \times \frac{\max R - r_i}{\max R - \min R}, & \text{if } f < 0 \text{ and } \max R \neq \min R \text{ or} \\ f \times \frac{r_i - \min R}{\max R - \min R}, & \text{if } f \geq 0 \text{ and } \max R \neq \min R \end{cases} \right\}, \tag{6.13}$$

where f_i is the feedback for the ith sequence, $r_i \in R$ is the model-based recognition belief for that sequence and $\min R$, $\max R$ the minimum and maximum model-based recognition beliefs among the alternative sequences. For the two sequences above the model-based belief is identical, -0.11, and therefore no discrimination is achieved. They both receive a feedback of $+1$.

The rationale behind this scaling is that positive examples with a higher model-based recognition belief are more likely to be the real positive ones and similarly for negative examples

with a low recognition belief. Since the feedback for each sequence is interpreted as an example weight and used in the refinement of the parameters for the event, this type of scaling minimises the incurred change to the model parameters, by giving less weight to examples which are near the parameter boundaries. A greedy alternative of this function would be to choose only one of the alternative sequences, e.g. the one with the highest/lowest positive/negative model-based recognition belief, and ignore the rest. The chosen function adopts a more cautious approach to feedback allocation, allowing all of the alternative sequences to be examined. An important characteristic of the chosen function is that it does not change the sign of feedback, it just scales it.

The last function that is used in feedback allocation is *comb-weight*, which calculates the weight of an example in a REST node, by combining beliefs of the sequences in which it participates. The result of the function is also used as feedback for the supporting example. If the examined REST node participates in a positive sequence, it is necessary for its recognition and should therefore be assigned positive feedback at least as high as the complete sequence. For this reason, *comb-weight* returns the maximum feedback:

$$\textit{comb-weight}(F) = \max F, \tag{6.14}$$

where F the set of feedback values for the sequences in which the node participates.

Figure 6.9, reproduces the example of Fig. 6.5, replacing the variables with numbers. Note that the feedback for the branch $\{X : (?, 5), W : (1, 2)\}$ of REST(Y) is 0.0, because it is the less negative of the two alternatives for the negative example $Y/\{\mathbf{3}, 8\}$, i.e., $Y(2, 10)$. As a result, example $W/\{\mathbf{1}, 2\}$, corresponding to occurrence $W(2, 4)$, receives 0.0 as feedback and will be ignored in the refinement of the parameters of W, because its classification is unknown.

6.6.4 Parameter refinement

The refinement algorithm used under partial supervision is the same as under full supervision. Once all the example weights have been calculated, a best-first search is performed which maximises a fitness measure, combining purity and proximity. The purity function is modified slightly to make use of the weighted examples. For instance, assume a solution, which covers sets P of positive and N of negative examples. Assume also that P_0 and N_0 are the sets of all

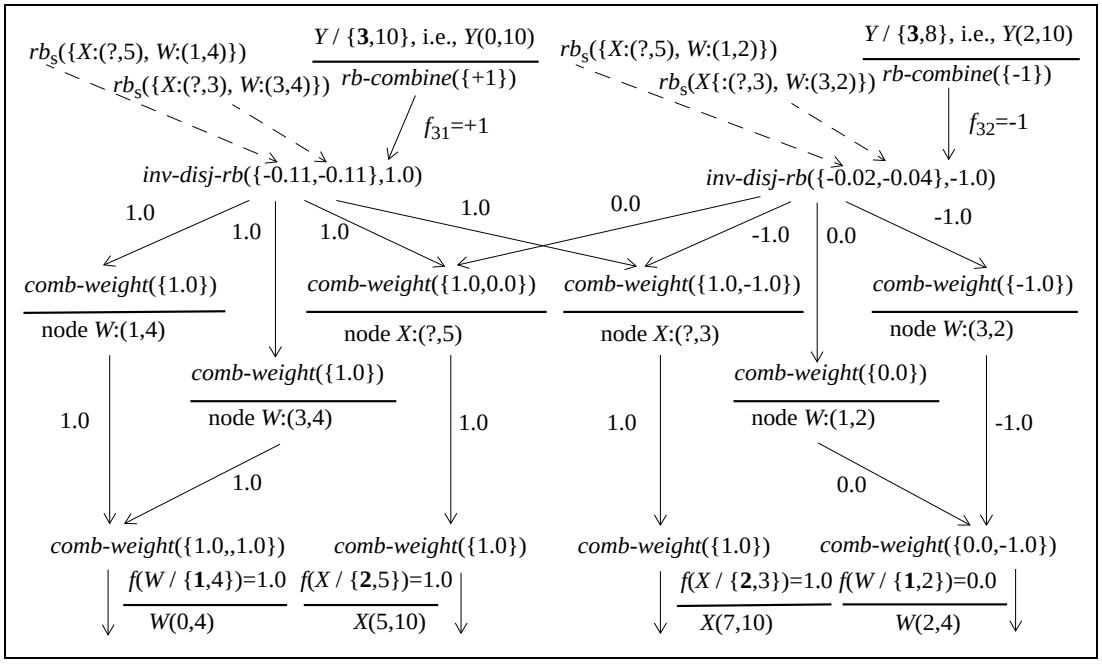


Figure 6.9: Calculated data-based belief values for $Y/\{\mathbf{3},8\}$, $Y(2,10)$ and $Y/\{\mathbf{3},10\}$, $Y(0,10)$.

positive and all negative examples. Then purity is defined as:

$$\textit{weighted-purity}(P,N,P_0,N_0) = \beta\frac{\sum_{i=1}^{n} w_i^+}{\sum_{j=1}^{k} w_j^+} + (1-\beta)\frac{\sum_{g=1}^{l}|w_g^-| - \sum_{h=1}^{m}|w_h^-|}{\sum_{g=1}^{l}|w_g^-|}, \quad (6.15)$$

where β is the user-defined *positive-to-negative ratio*, $w_i^+ \in P$, $w_j^+ \in P_0$ are positive weights and $w_h^- \in N$, $w_g^- \in N_0$ negative ones. When alternative positive examples are covered by the solution, the one with the highest weight is kept in P and P_0. The behaviour of the weighted purity function is very similar to that of the simple one, described in section 5.5.

The proximity heuristic under partial supervision is exactly the same as under full supervision. An alternative approach to incorporate a model-based bias would be to measure proximity implicitly, by including the model-based recognition belief in the weight of an example. This could be done simply, by including it in the input of the *comb-weight* function and treating it in the same way as the feedback values, i.e., choose the maximum amongst the model-based belief and the data-based ones. Then the purity function would incorporate model-based information and the proximity function would not be needed. However, this does not yield the desired result. Assume, for instance, two alternative positive examples p_1 and p_2, with weights

$w_1 > w_2$. Assume also two solutions h_1 and h_2. h_1 covers p_1 and is very close to the original model parameters, while h_2 covers both alternatives and requires a large change in the original parameters. All other things being equal, the model-based purity of both solutions will be the same, ignoring the large proximity difference of the two solutions. So, the model-based purity does not provide adequate discrimination between the solutions. Another important problem, is that *comb-weight* can turn negative examples into positive if their model-based recognition belief is positive. This is not the intended effect of the proximity bias.

As mentioned throughout the chapter, the handling of negative examples becomes particularly important under partial supervision. At the stage of parameter refinement, this issue is addressed by some additional pre-processing of the example lists, aiming to reduce the number of negative examples. The examples which are removed are the ones which do not support "covered" higher-order occurrences. What is meant by "covered" here is the set of occurrences covered by the *refined* parameter ranges for an event. Thus, if the selected parameter ranges for event Y exclude the negative occurrence $Y(2, 10)$, the supporting subevent occurrences, e.g. $W(2, 4)$, can be ignored. If they were not ignored they would be used as negative examples and the refinement algorithm would attempt to exclude them. However, this is not necessary, since $Y(2, 10)$ has already been excluded by the refined parameters of Y.

In addition to these negative examples which do not support "covered" higher-order event occurrences, a number of other event occurrences which are generated in the classification stage can be ignored. The use of parameter windows results in some intermediate event occurrences not being used in the recognition of higher-order ones and never be assigned any feedback. Since there is no classification information for the corresponding examples, they should be ignored in the refinement of the parameters for the intermediate event. Similarly, all examples which have received 0 feedback can be ignored.

6.7 Summary and critique

The refinement algorithm has been extended to operate under partial supervision. In this task, training information is provided for only the output nodes of the TCN and has to be allocated back to the intermediate nodes. In order to achieve this, first all event occurrences that can possibly be recognised in the input stream are recognised, ignoring the temporal constraints of the model. An initial recognition belief is assigned to each of the event occurrences and

the memory of the system is extended to store causal information. This information is then used by the feedback allocation algorithm, which updates the recognition beliefs for the event occurrences. The result of this process is stored as example weights in the RESTs of the events. The refinement algorithm has not been modified substantially. The only new feature is the new weighted purity heuristic.

The feedback allocation algorithm uses a number of heuristic functions. The choice of these functions has an important effect in the refinement process. The selected functions seem intuitively sensible, but lack sound theoretical basis. An alternative approach would be to implement a probabilistic belief updating system, the properties of which have been extensively studied. The limiting factors for the use of probabilistic rules is the lack of data and the large amount of uncertainty and interdependence in a TCN. In that sense, the system presented in this chapter takes an optimistic view, making decisions that are not statistically justified. The comparison of a probabilistic feedback allocation system with the one developed here is certainly of interest and will hopefully take place in the future.

An additional feature of the system presented here is the large number of heuristics. This is a cause of concern, since each of them is chosen individually, rather than using a global framework, e.g. a probabilistic one. The lack of such a framework can result in incorrect or inconsistent heuristics. This problem has been addressed by the use of explicit criteria and biases, which at least lead to a consistent system. The result is a set of simple and similar heuristic functions. Although they have a different name and are used at different stages of the process, the behaviour of most of the functions is either identical or inverse.

Finally, a practical problem which was mentioned in chapter 5 is that RESTs can become quite large. A method of pruning the REST has been introduced, which uses parameter windows. The problem becomes more serious under partial supervision, because of the large number of intermediate event occurrences that are recognised. This phenomenon is due to the lack of feedback for intermediate events and the adopted approach to generate all possibly recognised event occurrences. In order to reduce the storage requirements of the algorithm, an additional method of compression is introduced in the construction of the REST. The new type of REST is called *condensed* and allows a list of durations to be stored at each node, instead of a single duration value. Durations can be grouped together if they participate in alternative sequences of the same example. Duration lists have to be combined with lists of distances for

the children of each such node. Figure 6.10 presents the *condensed* REST(Y) of Fig. 6.3. The

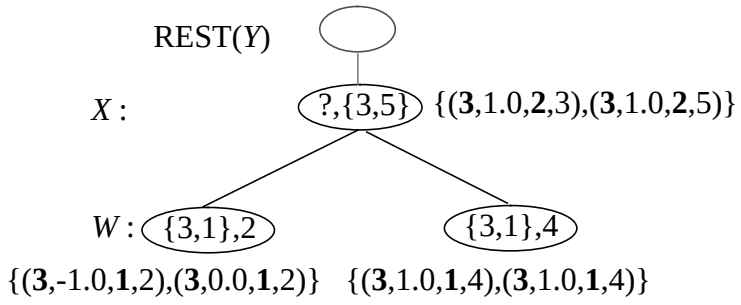


Figure 6.10: An example of a condensed REST. Each node contains a list of examples of the format: (**example**,weight,**subevent example**,subevent duration).

node at the level of subevent X contains now both possible durations, 3 and 5, for the supporting subevent and its children contain respective lists of distances. Note that each example in the example lists contains an extra element, corresponding to the duration of the supporting subevent occurrence. The type of REST examined so far is named *flat* REST. Condensed RESTs are only used for storage, at the moment, and they need to be flattened in the refinement process. The extension of the refinement algorithm to use the condensed RESTs directly could improve its computational and space efficiency.

Part IV

Experimental Results

Chapter 7

Theme analysis of humpback whale songs

During the breeding period, humpback whales sing a highly structured and complex song. This song has been studied extensively by marine biologists in the past 30 years. A large number of recordings have been made and analysed and general agreement has been reached about the basic elements and the dynamics of the song. The song of the humpback is of particular interest to this study, due to its temporal features and hierarchical structure. A small set of songs has been transcribed and coded to be used as a testbed for the methods presented in the previous chapters. The structure of the song has been modelled by a generic temporal classification network (TCN) and a simple initialisation method has been developed to set the temporal parameters of the model, using a subset of the data. Models initialised in this way are refined using the remaining data, less one song withheld for testing. The task of the classification model is to recognise the high-order components of the songs, called *themes*.

7.1 Song of the humpback whale

7.1.1 Basic properties of the song

The humpback whale, *megaptera novaeangliae*, is a baleen whale which can be found in most oceanic areas. It spends the summer season, May to November, feeding near the poles and migrates closer to the equator for breeding in the winter, December to April.[1] Humpback whales produce various sounds during both periods. However, during the migration and breeding seasons, unstructured vocalisations give way to a complex song. Most evidence suggests that

[1]For a textbook introduction in the life of humpback whales see [111].

this song is sung by male animals and has a similar functionality to bird song, i.e., successful mating of the singer [99, 100].

The song of the humpback is a pattern of sounds, which may be repeated several times in sequence. A sequence of songs is called a *song session*. The duration of a song varies from a few minutes to roughly half an hour, while a song session can last several hours. Singing is usually performed by lone individuals, who repeat the same song with very little variability. Two interesting properties of the song are that there is no distinguishable spacing between songs in a session and that the session may start at different points within a song. As a result, it is difficult to determine the beginning of each song and the conventions that are used are arbitrary.

Most of the above information is reported in [79] and [113], which were the first systematic studies of the song. The former study analysed recordings from the North Atlantic breeding area of Bermuda, between 1957 and 1971, while the latter used recordings from the Caribbean Sea, between 1969 and 1977. Payne and McVay [79] present an analysis of the song structure, which has been very influential for later research. Based on analysis of the spectrograms, frequency vs. time diagrams, of the songs, they proposed the following structure:

- A *song session* is a sequence of repeated *songs*.
- A *song* comprises a sequence of *themes*.
- A *theme* is a repetition of *phrases*.
- A *phrase* is a sequence of *units*.
- A *unit* is a sound which is continuous to the human acoustic perception and is usually considered the most elementary building block for the song. Units are sometimes decomposed into *subunits*, e.g. in the case of a pulsive unit.

In later research [37, 76], this structure has been slightly modified to give a more precise description of the song:

- A *phrase* is decomposed into *subphrases*.
- A *subphrase* is either a fixed sequence or a repetition of *units*.

Figure 7.1 gives a graphical view of this structure.

The basic unit sounds vary in many different ways:

- Their fundamental frequency varies between 30 and 2,500 Hz.
- Some units have a rich harmonic structure and others do not.

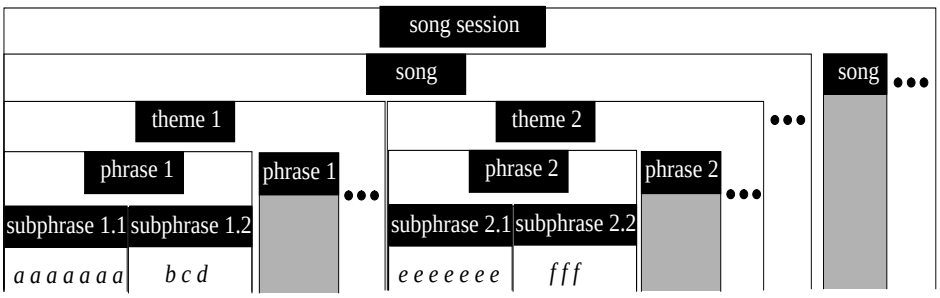


Figure 7.1: Overview of the humpback whale song. a, b, c, d, e, f are different types of unit.

- Frequency modulation differs between units: there are quick and slow upward and downward sweeps, flat frequency units and other more intricate frequency variations.
- Their duration varies from a few tenths of a second to about 8 seconds.
- The temporal distance between units is roughly in the same range as their duration.
- There are tonal and pulsive units.

The variation between units and the predictable structure of the song make the analysis of the song into its subcomponents a manageable task for human experts. However, one should not underestimate ambient noise, echoes and other interference, which can affect significantly the quality of underwater recordings. The aim of the work presented in this chapter is to automate this analysis, with the use of an expert model of the song. Refinement of this model, in the form presented in the previous chapters, is of particular interest, because the duration of and temporal distance between unit utterances can vary significantly, especially between individuals [39, 30].

7.1.2 Peculiarities of the song

Despite its apparent predictability, the humpback whale song has a number of very interesting features, which complicate its analysis. An early observation was that essentially the same song is sung by all whales in the same ocean basin, but the songs in different ocean basins differ substantially [112, 78]. The differences are in terms of song structure, i.e., the types of unit that are used and the way in which they are combined into phrases and themes. Payne and Guinee [78] note that the differences in unit types are smaller than in song composition, suggesting that there may be a characteristic sound repertoire for the animal.

Searching for a justification of these differences between songs, another interesting feature was discovered: the structure of the song, sung by a population, or stock, of whales, is continuously changing. This explains the development of different songs by acoustically separated populations. Several short and long-term studies of the gradual change of the song have been published [75, 76, 37]. The first interesting conclusion of these studies is that the song changes during the breeding season. There have been very few recordings of songs in the feeding grounds, suggesting that the whales do not sing in this period [55, 57]. This observation explains why the song remains in a fairly stable state during that period. Changes to the song are introduced by individuals [37] and are either adopted by the rest of the population or they disappear again after a few months [78]. In the short term, i.e., a few months, changes to the song are made at the unit and phrase level. Examples of this are (from [78]):

- Changes in the fundamental frequency of units.
- Addition or removal of units.
- Changes in the duration of units.
- Changes in the number of repeated units in a phrase.

In the longer term, complete themes can disappear and new ones take their place.

Other studies have looked at the composition of the song at a particular point in time, attempting a different classification of its components. In [37] subphrases are identified which appear in more than one theme. Examples of this phenomenon are presented in section 7.4. These are called *multi-theme* subphrases and it has been suggested that they help in remembering the complex song structure. Additionally, the authors distinguish between *shifting* and *static* phrases and units. The former type corresponds to phrases and units which gradually change in the progress of a song and presents a particular difficulty in the modelling of the song. Multi-theme phrases are always static, i.e., of a fixed structure. Finally, there is a special type of theme, which is called *transition theme*. A transition theme consists of a single phrase, which combines material from different themes, usually one subphrase from the preceding and one from the following theme.

Another complication of the humpback whale song is the presence of *aberrant* songs within a singing period. An extensive study of aberrant songs is presented in [33], where a song is classified as aberrant if it misses a *fundamental* theme. A *fundamental* theme is one which appears in 90% of the songs in a period. The author identified 14 aberrant song sessions, out

of 123 that he examined, and separated them into three types:[2]

A. Abridging the song to start again from a particular theme (4 sessions).

B. Arbitrary mixing of material (3 sessions).

C. Short-term song change (5 sessions, concentrated in a period of a few days).

In addition to these top-level structural aberrations, minor deviations from the main song structure are also possible (see section 7.3).

7.1.3 Automatic classification of marine mammal sounds

Work on automatic processing of marine mammal sounds has focused on the low, signal processing level, where the goal usually is *unit recognition*. Due to the similarity of the task to phoneme recognition in speech processing, several methods from this area have been applied to the problem [39, 13, 15, 58]. However, the special properties of the underwater acoustic signal, i.e., ambient noise, uncontrolled utterance sequences, etc., have impeded the success of these approaches.

The family of whales which is most often studied in marine mammal sound processing is *odontocetes*, i.e., toothed whales, especially dolphins, due to the abundance of recorded data. A recent example of this work is presented in [96], where various types of Artificial Neural Network (ANN) have been compared on the task of classifying three types of bottlenose dolphin sounds. An attempt was also made to extend this task to higher-order symbolic processing, using a method based on fuzzy sets [96, 3]. The goal of the extended system was still low-level sound classification, which is called *unit classification* here.

ANNs have also been applied recently to the task of unit classification for baleen whale songs, e.g. bowhead whales [81]. Like the work presented in [96], this work is image-based, i.e., processing the digitised image of a spectrogram. Some of the problems which have been noted with this approach are:

- Difficulty in isolating the sound segment in the digitised image.
- Need to centre the samples manually.

Due to these problems, the proposed unit classification process is still very laborious.

Finally, there have also been attempts to apply clustering, or unsupervised learning, techniques to the digitised images of humpback whale units. The first such attempt [11] has not

[2] The two missing sessions were too short to be analysed.

been very successful, generating an excessively large number of unit types and having a high misclassification rate on unseen data. More recently the method of self-organising maps has been applied to the problem with more success [108]. The authors of this work have observed the emergence of clear patterns in the induced map, which can be used to discriminate between unit types in the song. The association of these unit types to higher-order structural components of the song was not attempted.

The methods presented in this thesis do not deal with low-level event recognition and can therefore not be used for unit recognition. The rest of this chapter concentrates on the use of the output of a unit classifier to perform higher-order analysis of the song.

7.2 Transcription of the songs

In the last 40 years, the interest in monitoring whales for conservation purposes has been ever increasing. This process involves detailed research on different aspects of the animals' behaviour, including vocalisations. As a result, large libraries of marine mammal sounds have been created, archiving recordings in various formats.[3] The songs acquired for this thesis were kindly provided by the Whale Conservation Institute (WCI),[4] in the form of photocopies of spectrograms. The reason for choosing the paper format was that an automatic unit classifier was not available and spectrograms are the natural choice for manual processing. The original spectrograms were produced by photographing the output of a spectral analyser, using a 35mm oscilloscope camera, on a continuous strip of 35mm photographic paper. The paper strip moved at a rate of 0.5cm/sec. The developed strips displayed frequency on a linear scale, in the range 40 to 2,500 Hz, against time. The strips were cut to pieces of roughly 60cm length and glued on A1-size paper. Each such *spectrogram log* contained at most one song session, but a session might require more than one log.[5]

The acquired spectrograms correspond to the complete set of 132 sessions recorded in the period between December 1977 and April 1978 off the coast of Maui, in the Hawaiian islands. This is the largest set of recordings available for a single period in WCI. Recordings from a single breeding season were preferred, in order to minimise the effect of song change over time.

[3] Unfortunately, the number of digitised recordings is still limited.
[4] Contact addresses for WCI and other sources for marine mammal sounds are provided in appendix F.
[5] More details about the machinery used and the logging process can be found in [76].

An additional reason for choosing that particular period is that a very detailed analysis of the song in that period has been published in [76]. In the process of analysing the songs, themes and phrases were clearly identified and labelled on most of the spectrograms.

The quality of the acquired photocopies of spectrograms suffered in many respects:

- There are cases where ambient noise and other interference in the original spectrogram, made impossible or very difficult the analysis of a complete song from the session (see example in Fig. 7.2).

Figure 7.2: Example of a noisy piece of recording.

- Almost all of the copies were missing a small part to the left or the right of the log.
- There were photocopy distortions in most of the copies.

However, the most important problem in the transcription of the songs was the size of the job. The manual transcription process consists of measurement, with the use of a ruler, of the begin and end points of all the units in a song. A typical song consists of roughly 250 units and its careful transcription takes about 4 hours. For that reason 17 sessions were selected out of the 132, on the basis of quality, and out of these the 10 first, chronologically, were transcribed.

As there is very little variability between songs in the same session, only one song was selected and transcribed from each session, with the exception of the first session. In the session of 8/12/77, one and a half songs were transcribed. The quantity of the transcribed material is presented in table 7.1. In an attempt to minimise manual transcription errors, consistency checks were performed on several properties of the encoded data, e.g. duration of units, distance between them, expected unit sequences, etc.

Note that the manual transcription process results in effectively noise-free data, since ambient noise and other interference are not coded. The issue of unit misclassification is discussed in the section 7.4.

song number	recording date	number of units	number of phrases	number of themes
1	8/12/77	458+220=678	52+25=77	14
2	10/12/77	209	46	9
3	10/12/77	408	44	9
4	19/1/78	410	21	7
5	19/1/78	247	21	7
6	25/1/78	166	32	7
7	29/1/78	271	11	8
8	7/2/78	125	18	8
9	26/2/78	181	26	11
10	9/3/78	227	20	8
Total		2922	316	88

Table 7.1: Size of the ten songs.

7.3 Generic classification model

The event recognition task examined in this chapter focuses on the themes of the humpback whale song. A generic TCN model is used which describes the structure of individual themes in terms of their subcomponents. The model has been constructed on the basis of song descriptions appearing in the literature and the spectrogram logs, on which themes, phrases and subphrases are marked. This section provides a detailed top-down description of the model, separating the song into themes, phrases, subphrases and units. This is in accordance to the descriptions found in the literature. Moreover, some rare aberrancies are described, which have been discovered during the transcription of the data and do not appear in the literature.

It should be stressed that the aim of the experiments is the evaluation of the methods proposed in this thesis, which refine the temporal parameters of a TCN model. The structural components of the model, including the number of repetitions in the definition of repeating events, *cannot* be acquired from the data automatically. The automatic acquisition of model structure from the data would be desirable for the song of the humpback whale, due to the aberrancies and variations appearing in the song. Since this type of learning is not dealt with in the thesis, the variations and aberrancies which have been encountered are included explicitly in the model. The model description presented in the rest of this section pays particular attention to this issue.

As described in [76], the song of the humpback whale in the recording period, consisted

of 8 regular themes and one transition theme. The regular themes have been given numeric labels 1 to 9, missing 3,[6] while the transition theme appears between themes 2 and 4. In addition to these, a transition theme between themes 5 and 6 has been encountered during the transcription of the ten songs used here. This an unusual case of transition, because it combines the beginning of theme T_6 with the ending of theme T_5, rather than the beginning of T_5 with the ending of T_6. Table 7.2 lists the themes that are found in each of the ten songs. Themes are labelled by T_i, where the subscript corresponds to the labels given to them in [76].

song number	T_1	T_2	$T_{(2-4)}$	T_4	T_5	$T_{(5-6)}$	T_6	T_7	T_8	T_9
1	2	2	2	2	2	0	1	1	1	1
2	1	1	1	1	1	0	1	1	1	1
3	1	1	1	1	1	0	1	1	1	1
4	1	1	0	0	1	1	1	0	1	1
5	1	1	0	1	1	0	1	0	1	1
6	1	1	0	1	1	0	1	0	1	1
7	1	1	1	1	1	0	1	0	1	1
8	1	1	0	1	1	1	1	0	1	1
9	1	1	1	1	1	0	1	0	3	2
10	1	1	1	1	1	0	1	0	1	1

Table 7.2: Themes in each of the ten songs.

As can be seen in Tab. 7.2 the composition of the ten songs in terms of the themes varies significantly. There is a number of fundamental themes, e.g. T_1, T_2, which appear in most songs and others which only appear in a small number of songs. More specifically:

- T_7 is dropped midway through the season, as noted also in [76].
- Song 4 is missing $T_{(2-4)}$ and T_4, due to the quality of the recording.
- Songs 4 and 8 are the only ones having the transition theme $T_{(5-6)}$.
- Song 9 presents another interesting phenomenon which was noted in [76]: the alternation between T_8 and T_9. The following pattern of themes is sung: $T_8 - T_9 - T_8 - T_9 - T_8$.
- Song 1 is actually one and a half songs, the second part stopping at T_5, due to the bad quality of the recording.

Since none of the themes is omitted from the training and test data for the experiment, the variation in the song composition is an important aspect of the experiments. By varying the choice

[6] Theme 3 existed in the previous season, which is also examined in [76].

of training and test songs, the material available for training and testing can vary significantly. This is issue is discussed in more detail in chapter 8.

The rest of this section examines in detail each of the ten themes. The labelling notation is as follows:

Themes: T_i

Phrase of theme T_i: P_i

Subphrases of phrase P_i: SP_{ij}

Units: sequence of small italic letters ($a, b, c, \ldots$).

Note that the units in different themes are labelled with different letters, even if they look the same on the spectrogram. Henceforth, this labelling scheme will be referred to as *context-sensitive unit classification*. Section 7.4 examines the problems of this labelling approach and proposes an alternative. In the following theme descriptions the qualitative descriptors *low*, *medium* and *high* frequency, relative to the range covered in the spectrograms, are used. The three discrete values correspond to three, roughly equally-sized, frequency ranges. The way in which these are determined is discussed in section 7.4.

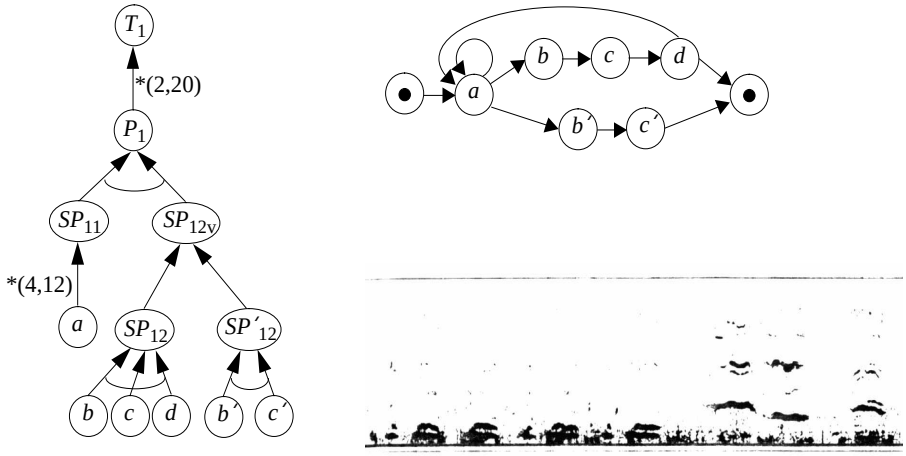
Theme T_1 is a repetition of phrase P_1, which is composed of two subphrases SP_{11} and SP_{12}. The first of the two subphrases is a repetition of a low-, flat-frequency unit, a, of alternatively short and long duration. The order in which the short- and long-duration a units are sung has been given particular attention in [76], because it used to be random in the previous season. In the season examined here the order of short and long a units is less random, but still not completely predictable. In the model presented here, the choice has been made not to distinguish between the two types of a. The reason for this is that the duration of units is a property which can be expressed explicitly in the model and therefore does not need to be taken into account in the labelling of the units. The second subphrase is composed of three units: b, c, d, which are of medium-, mainly flat frequency. One aberrant pattern of SP_{12}, appears in song 7, where there are only two units: b', which is similar to b, and c', which looks like a hybrid between c and the first unit in theme T_2. This phenomenon can be seen as a special type of a shifting unit, where there is just one connecting unit between two different types. As mentioned above such aberrant cases are included explicitly in the model. In this case, the aberrancy is modelled as a second subphrase, SP'_{12}. Figure 7.3(a) presents graphically the definition of T_1,

including examples of the three subphrases. In addition to the TCN representation, a similar state-transition-network (STN) representation and a spectrogram sample of the typical phrase for the theme are provided.

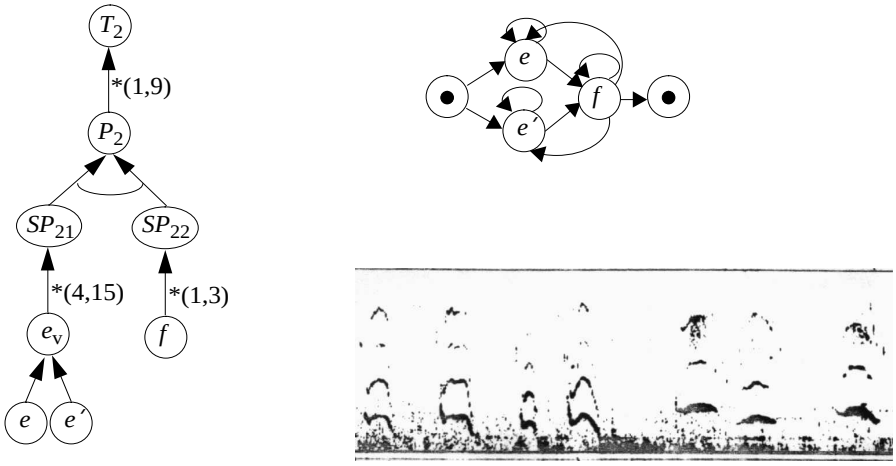
Theme T_2 consists of repeated P_2 phrases, which are made of subphrases SP_{21} and SP_{22}. The first subphrase is a repetition of unit e, which starts with a downward sweep from a high frequency, remains at a medium, flat frequency for most of its duration and decreases again at the end. In song 4, a different type of unit, e', is used in SP_{21}, which is very short, at a frequency level roughly equal to the flat portion of e. The frequency of e' is rapidly rising and falling again. This type of unit has been used in theme T_2 in the previous season. SP_{22} is a repetition of 1 to 3 medium, flat-frequency units, f. Figure 7.3(b) describes T_2. Comparing the spectrograms in Fig. 7.3(a) and 7.3(b), it becomes clear that the last three units of the two phrases are very similar, i.e., SP_{12} is similar to SP_{22}. Despite this fact, the units in SP_{12} and SP_{22} are labelled differently by the hierarchical decomposition approach adopted here. This is a problem with the context-sensitive labelling scheme and is discussed in detail in section 7.4.

Theme T_4 is a repetition of P_4 phrases, containing subphrases SP_{41} and SP_{42}. The first subphrase is usually a combination of two units: g, h, of which g is flat, medium-frequency and h is similar, but very short and with a drop in frequency at the end. The usual pattern is $g - h - g$, but there are cases, where h and possibly one of the gs is missing. This aberrant phenomenon is captured by SP'_{41}, which is defined as one or two gs. SP_{42} is a repetition of i units, which are of low, but rising frequency. Figure 7.4(a) presents this theme.

Theme T_5 consists of repeated P_5 phrases, which are made of three subphrases: SP_{51}, SP_{52} and SP_{53}. The first subphrase is a combination of two units: j, k in the pattern: $j - k - j - k$. They are almost identical to units g, h of P_4. The second subphrase is a single unit, l, of long, flat and medium frequency. This subphrase is only modelled implicitly as part of P_5, i.e., SP_{52} does not appear in the model, because of its single-unit structure. SP_{53} is a repetition of units m, which are very short and of low and decreasing frequency. Figure 7.4(b) shows the structure of this theme.



(a) Theme T_1



(b) Theme T_2

Figure 7.3: Theme descriptions (T_1, T_2). TCN, STN and spectrogram sample. The notation SP_{12v} is used for a disjunction of two alternative models of SP_{12}. The upper and lower bound on the number of repetitions in repeating events is shown as $*(x, y)$ beside the corresponding links. Conjunctive events are denoted by an arc connecting the incoming links, similar to the AND notation in AND/OR trees. The STN representation ignores the silence periods between unit utterances. The spectrogram samples correspond to roughly 5 seconds of recording each.

Theme T_6 in the songs examined here, consisted always of one phrase, which is not explicitly modelled. There are two subphrases of the phrase: SP_{61} and SP_{62}, the first of which consists of repeated units n. The units of this subphrase have been examined in detail in [76], because they changed from the previous season. No major variation has been found in the songs examined here. Unit n is a short upward sweep, starting from a low frequency. SP_{62} consists of two flat, low-frequency units, o, p. This subphrase is similar to the second subphrase in themes T_7, T_8 and T_9 and has been described as a rhythmic phrase in [38], used in remembering the song structure. Figure 7.4(c) describes T_6.

Theme T_7 has only one difference from T_6: the unit of the first phrase, q, is pulsive, rather than tonal. Theme T_7 disappears in the later songs during the season. Figure 7.5(a) shows the structure of T_7.

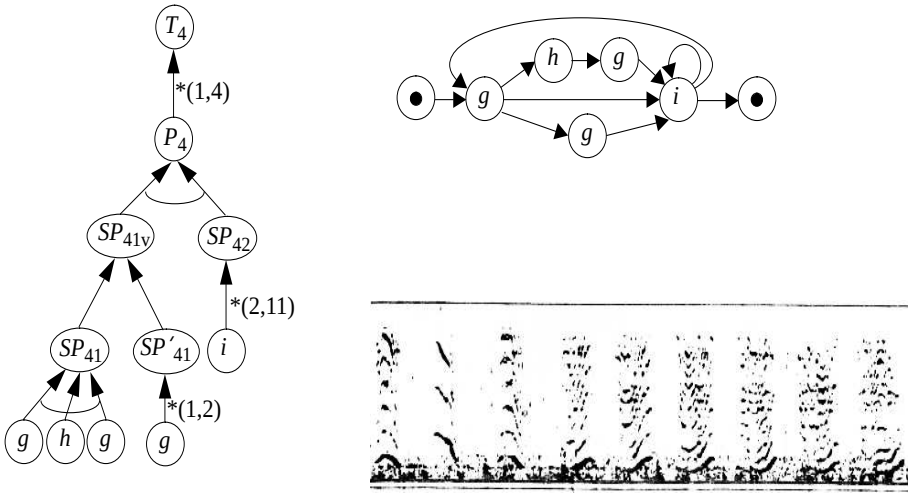
Theme T_8 is a repetition of P_8 phrases, consisting of subphrases SP_{81} and SP_{82}. The first subphrase is a fairly arbitrary mix of units: t, t', t'', modelled by the disjunctive "*super-unit*" t_v. The first, t, is of a rising frequency, the second, t', of a falling one and the third, t'', is an upsweep, followed by a downsweep at high frequency. The last type of unit is quite rare. The second subphrase is the same as SP_{62}, but at a lower frequency. Figure 7.5(b) presents the definition of the theme.

Theme T_9 is similar to T_8, but the first subphrase, SP_{91}, is an ordered pattern. It combines two units: w and x, in the pattern: $w - x - w - x$. Unit w is flat and usually at a low frequency. Near the end of the season however its frequency rose. The second unit has a falling frequency, similar to t'. Figure 7.5(c) presents this theme.

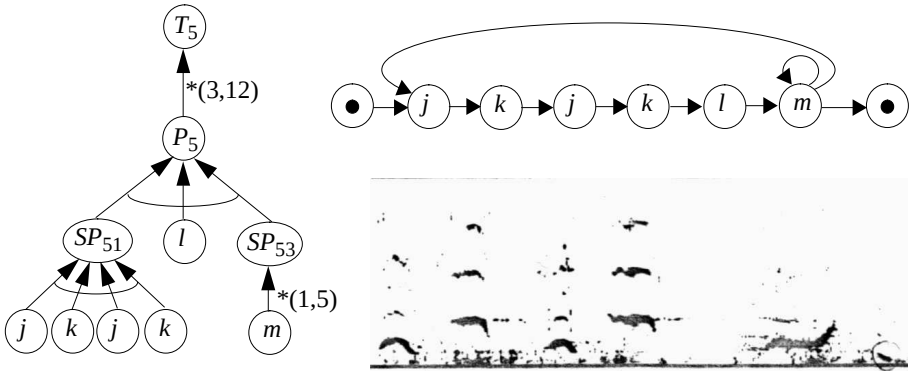
Theme $T_{(2-4)}$ is a common transition theme, composed of subphrases SP_{21} and SP_{42}. As mentioned above transition themes contain a single phrase, occurring only once. Figure 7.6(a) shows an example spectrogram of $T_{(2-4)}$.

Theme $T_{(5-6)}$ has been seen in songs 4 and 8 and is like P_5, but instead of SP_{51}, it contains SP_{61}. Figure 7.6(b) reproduces the spectrogram of this theme.

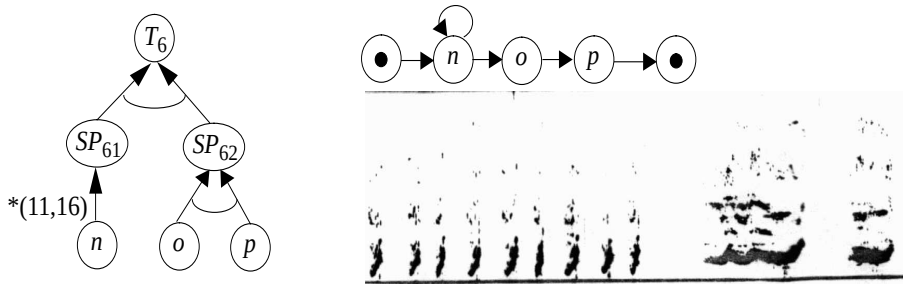
Using the structure of the song as described above, Fig. 7.7 shows the distribution of phrase



(a) Theme T_4

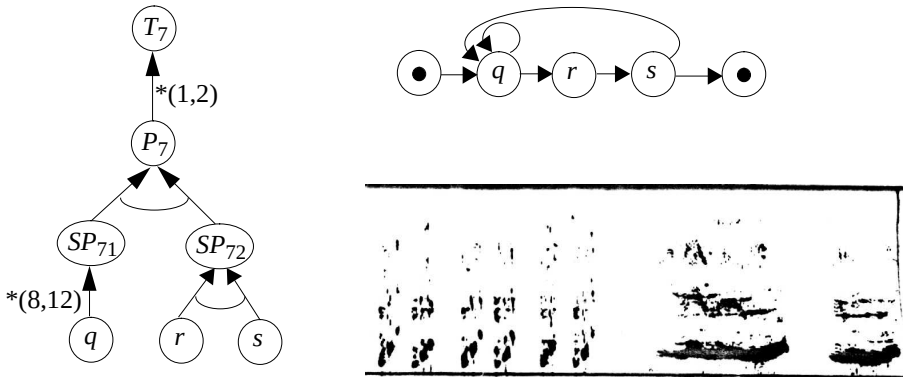


(b) Theme T_5

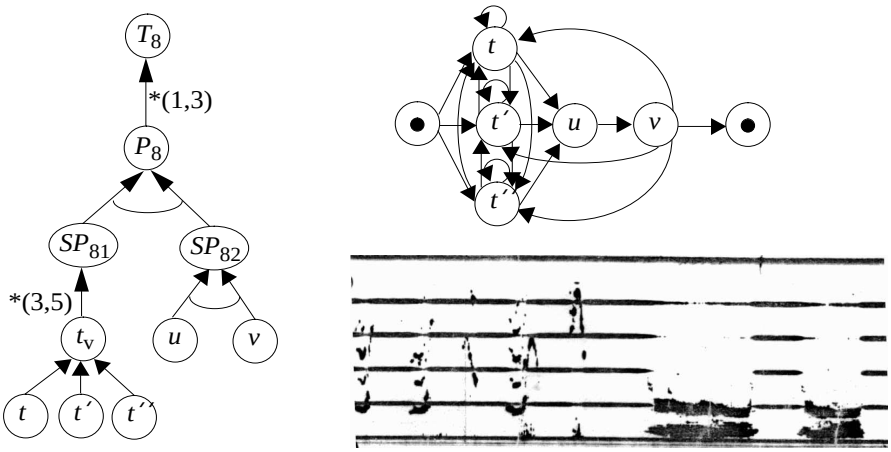


(c) Theme T_6

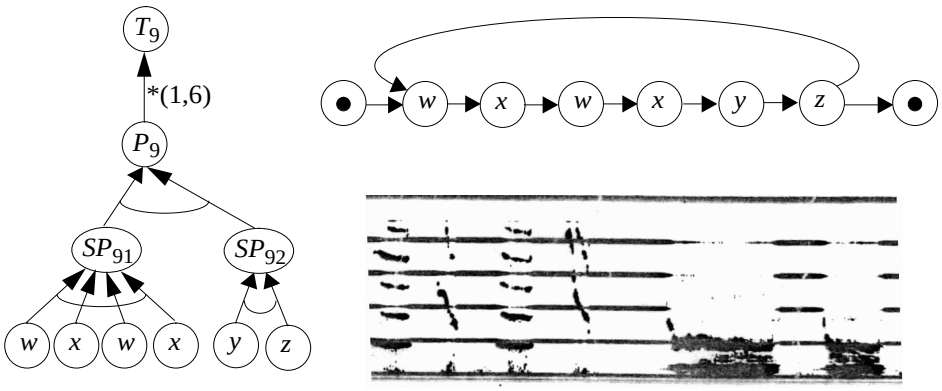
Figure 7.4: Theme descriptions (T_4, T_5, T_6). The same conventions as in Fig. 7.3 are used.



(a) Theme T_7

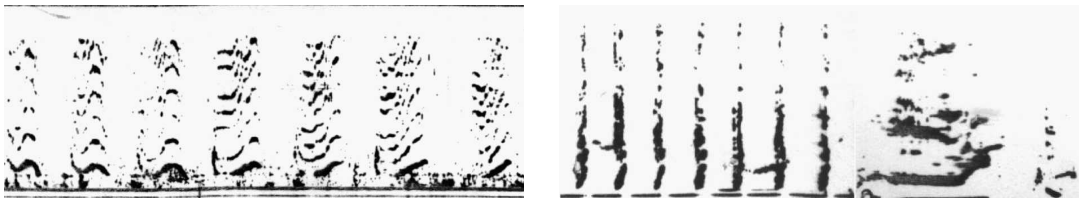


(b) Theme T_8



(c) Theme T_9

Figure 7.5: Theme descriptions (T_7, T_8, T_9). The same conventions as in Fig. 7.3 are used.



(a) Theme $T_{(2-4)}$ (b) Theme $T_{(5-6)}$

Figure 7.6: Examples of transition themes.

and unit occurrences in the ten songs. Some units clearly occur more often than others. In particular units a and e are very common, because they are:

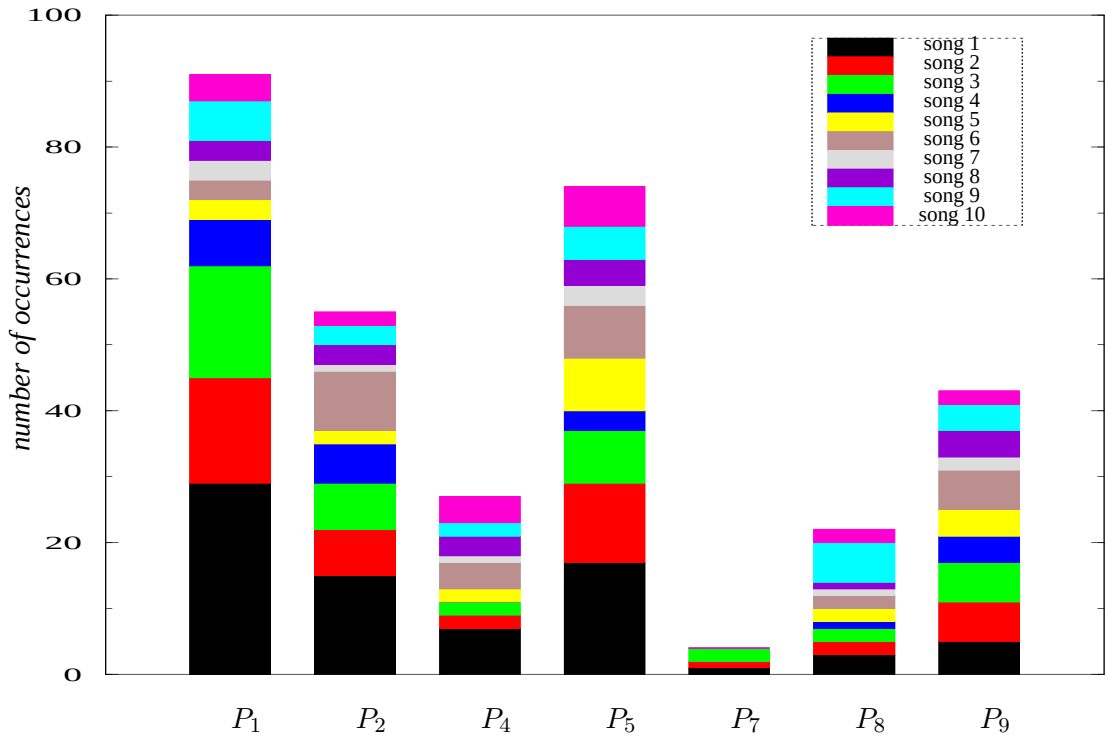
- parts of long repeating subphrases and
- the phrases in which they belong are also repeated many times, Fig. 7.7(a).

The units of phrase P_5, do not occur as often, i.e., j, k, l, m are not as many, despite the fact that P_5 is a highly repeated phrase. The reason for this is that the subphrases of P_5 are not repeating sequences. On the other end of the scale, some units occur only in one or two songs, e.g. b', e', etc. It is impossible to learn the parameters of the corresponding subphrases, unless these particular songs are examined.

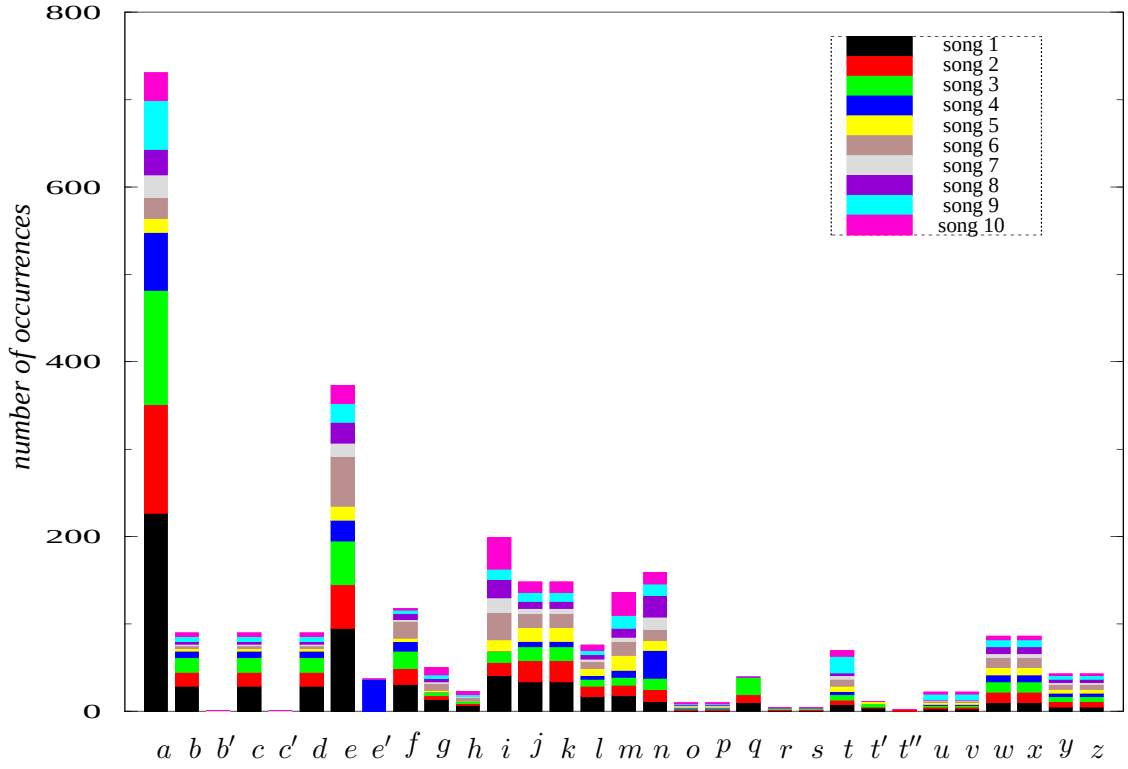
7.4 Unit classification

The model described above makes use of the context-sensitive labelling scheme, which provides a top-down, part-whole classification of the entities in the model, i.e., the events. This scheme is useful for describing the supporting relations in the model, but does not correspond to real differences in the labelled units – at least to the extent that these can be derived from the spectrograms. Clear examples of this phenomenon are the units of SP_{12} and SP_{22}, as mentioned above, and the second subphrases of T_8 and T_9. A context-free unit classification system, which made use of only the features that can be extracted from the spectrogram, would not be able to distinguish between these units. Thus, the context-sensitive labelling scheme contradicts the bottom-up design of the two-stage classification system, which was proposed in chapter 3.

In an attempt to resolve this contradiction and make the problem more realistic, this section



(a) Distribution of phrases.



(b) Distribution of units.

Figure 7.7: Phrases and units in the ten songs.

describes a method of clustering units into more meaningful unit types. The proposed method uses a simple *clustering rule*, which is a logical combination of qualitative criteria. This rule is a mapping of the 31 unit labels used above to a smaller set of unit types. The mapping is many-to-one, i.e., a context-sensitive label is mapped *uniquely* to a context-free type. The clustering rule makes use of only two simple criteria:

Qualitative shape description. The shape of each unit was classified into one of four groups: *flat*, *increasing*, *decreasing* and *inflective*, according to whether the main part of the sound was of flat, increasing, decreasing or of more complex frequency modulation. Most of the units fall into the first category, having only minor fluctuations in frequency. The remaining units are classified as:

increasing: i, n, q, t,

decreasing: m, t', x,

inflective: c', e, e', t''.

The level of the fundamental frequency. This can take the values: *low*, *medium* and *high*. The threshold values for the three categories are approximately: 300Hz and 600Hz. The process by which these values have been selected is as follows:

- For each of the songs, the frequency range for each context-sensitive unit has been determined approximately. The table in appendix G provides the resulting ranges.
- The midpoint of each range has been plotted for each unit in each song. The thresholds were manually selected in a way which puts most of the points for each unit in the same group. For the frequency-modulating units little extra separation was achieved and the unit types were easily separable:

  inflective-medium (type D): c', e, e',

  inflective-high (type E): t'',

  increasing-medium (type F): i, n, q, t,

  decreasing-low (type G): m,

  decreasing-high (type H): t', x.

  The flat-frequency units on the other hand had significant overlap in their frequencies, even when examining only the midpoint of the ranges. Figure 7.8

shows the frequency midpoints for all flat units and the selected thresholds. The thresholds do not provide perfect discrimination between the three unit types, but maximise the purity of each group. Each unit is classified into the group which contains the majority of its frequency midpoints. For example, units y, z, are considered to be low-frequency units, because 60% of their frequency midpoints lie below or on the 300Hz threshold. The resulting unit groups are:

flat-low (type A): a, u, v, y, z,

flat-medium (type B): $c, g, j, l, o, p, r, s, w$,

flat-high (type C): b, b', d, f, h, k.

Thus, there are now 8 unit types, labelled by capital letters in the above lists.

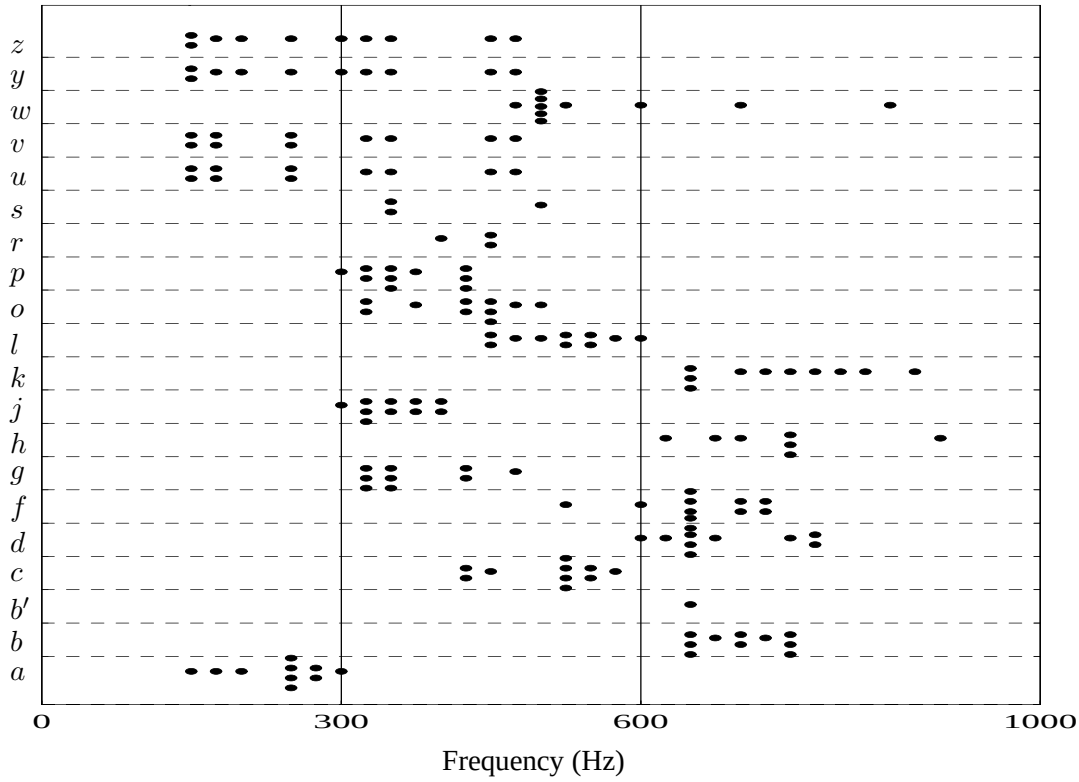


Figure 7.8: Midpoints of the frequency range for each unit in each song. Each dot represents the midpoint of the range which roughly covers the frequency of the corresponding unit in each song. For each unit there are as many dots, as the number of songs in which the unit occurs.

Clearly, the many-to-one mapping between units and unit types *does not* remove completely the context-sensitivity of the unit labelling scheme, but it *does* make the problem more

realistic. In order to achieve context-free unit classification, one would need to apply the clustering rule directly on the spectrograms and acquire a new classification of the songs' unit occurrences. This was not done here for the following reasons:

- The large number of units in the songs, requires a long processing time for manual re-classification.
- The small scale of the spectrograms, allows only approximate measurement of frequency information.
- The quality of the spectrograms causes further measurement problems.
- The clustering rule is oversimplified. A realistic unit-classification system, operating directly on the recorded songs, would take more information into account than this simple rule, e.g. number of inflection points and harmonic structure.

Since unit classification is not in the scope of the thesis, it was decided that the uniform re-labelling of units, according to the suggested mapping, is sufficient. Moreover, it is expected that the chosen unit types would be separable by an automatic clustering system, which has access to more accurate feature measurements on the sounds. The basis for this claim is that the chosen clusters are broad and differ in ways which are not captured by the simple clustering rule, e.g. harmonic structure, pulsive vs. tonal units.

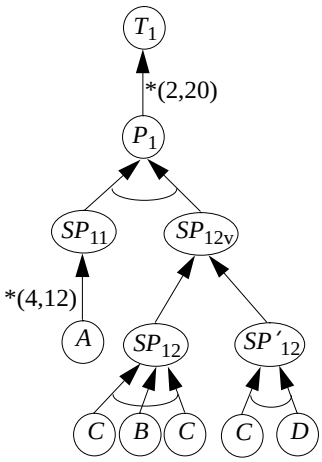
For the most part, the new labelling scheme does not affect the structure of the model. However, there are two important exceptions to this:

Theme $T_{6/7}$: The clustering rule does not examine whether a unit is tonal or pulsive. This is because it is difficult to extract this information from the spectrogram. As a result, the definition of T_6 and T_7 becomes identical. These definitions have been replaced by a new theme $T_{6/7}$, which has the same structure as T_7.

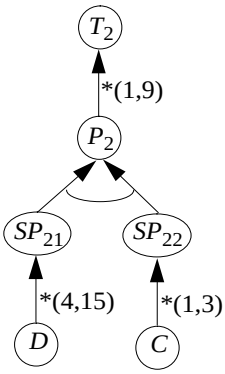
Subphrase $SP_{82/92}$: As mentioned in the previous section, subphrases SP_{82} and SP_{92} are identical. For this reason, they have also been merged into a common subphrase.

Figure 7.9 describes the eight themes using the unit types, A to H. In addition to the few structural changes, the new labelling scheme introduces a significant amount of overlap in the definition of subphrases. The following are some cases, where this has an important effect:

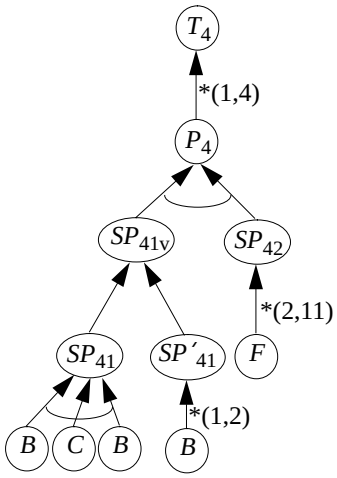
- $c', e \in D$: This reflects the shift of the abnormal subphrase SP'_{12} towards the next theme, T_2. The effect of this in the only song where SP'_{12} appears, song 7, is that the



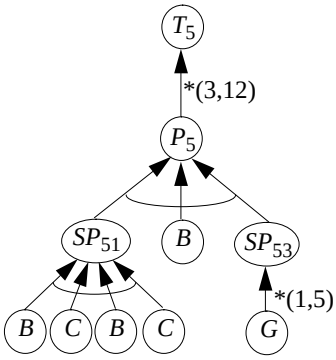
(a) Theme T_1.



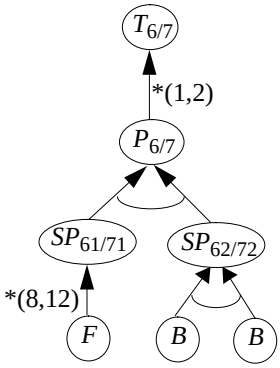
(b) Theme T_2.



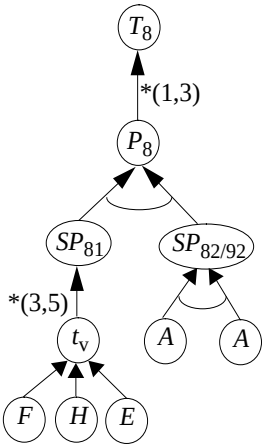
(c) Theme T_4.



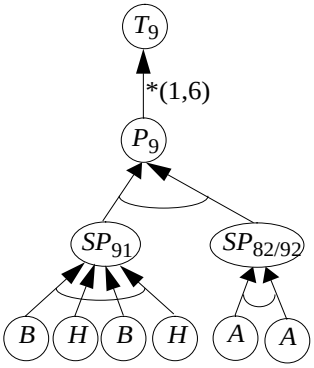
(d) Theme T_5.



(e) Theme $T_{6/7}$.



(f) Theme T_8.



(g) Theme T_9.

Figure 7.9: Theme descriptions using unit types.

recognised SP_{21} starts earlier and overlaps with SP'_{12}.

- SP_{22} fires for a single C: The effect of this is that SP_{22} can be recognised, incorrectly, in SP_{11}, SP'_{12}, SP_{41} and SP_{51}, all of which contain a C-type unit in their definition. To a certain extent this error will be avoided during classification by the duration constraint imposed on C, by different subphrases. During learning however the temporal parameters are assumed to be unknown.
- SP'_{41} fires for a single B: This is a similar phenomenon, but a more serious problem, since B is a more commonly used unit-type.
- SP_{41} is subsumed by SP_{51}: This reflects the similarity of the two subphrases.
- SP_{42} is similar to $SP_{61/71}$: Again this is a problem for learning, but not for classification, since the duration constraint on F is different for the two subphrases.

The effect of the increased uncertainty, introduced in the model by the context-free unit labelling scheme, is further examined in chapter 8.

7.5 Automatic model initialisation

The biological literature does not provide a detailed study of the temporal properties of the humpback whale song. The only available information is a rough estimate of the duration of a unit, a phrase and a theme [76], i.e., no specific information about individual units, phrases and themes. In order to apply the refinement algorithms to the model, the temporal parameters need to be assigned initial values, which are reasonable, but not accurate, estimates of the desired parameter values. The solution adopted here to this problem is to devise a simple method, which sets automatically the parameters, to *cover* a subset of the training set. For instance, if 9 out of the 10 songs are chosen for training, a subset of these, e.g. 5, are used for initialising the temporal parameters and the remaining, i.e., 4, are used for refinement. The initialisation method sets the parameters in a way that all the components of each song in the subset, i.e., the 5 songs, are correctly recognised. This approach can be seen as a primitive, problem-specific type of learning, since it extracts the parameter values from the data. This section describes how this can be done for the context-sensitive and context-free models.

7.5.1 Context-sensitive model

In the case of context-sensitive unit labels, the mapping between units and subphrases is roughly many-to-one, i.e., each unit can only participate in the definition of a single subphrase. Exceptions to this are the rare variants of some subphrases, e.g. SP'_{41} uses unit g, which is also in SP_{41}. The many-to-one mapping in the context-sensitive model simplifies the task of automatically setting the temporal parameters:

- Each duration range for a unit, needs to be large enough to exactly cover the duration of *all* occurrences of the unit in the examined songs. For example, the duration constraint for a, in the definition of SP_{11}, will have a minimum value equal to the shortest a and a maximum value equal to the longest a in the songs, the reason being that a is only used in SP_{11}. Note that for units which appear in more than one subphrase definitions, e.g. g, this will lead to overgeneralisation of the duration ranges, since all relevant subphrase definitions, SP_{41} and SP'_{41} in the case of g, are updated simultaneously. A similar inaccuracy occurs in units which are used more than once in the definition of a subphrase, e.g. units w and x in SP_{91}. The duration constraints for both supporting links are updated simultaneously, leading to overgeneralisation. The effect of this on the initialised model is discussed again in chapter 8.
- Similarly, distance ranges for subphrases can be uniquely determined by pairs of units. Again some inaccuracy will result from the fact that identical pairs may appear more than once in a subphrase definition, e.g. SP_{51} and SP_{91}.
- The temporal parameters in phrase and theme definitions can be derived in the same way, since subphrase and phrase labels are also context-sensitive.

One issue that arises with this method is how to set the parameters for events which do not occur in the songs used for the initialisation of the model. The solution adopted here is to establish surrogate events and use their parameters. For instance, SP_{41} is set to be a surrogate for SP'_{41}.

Applying the parameter-setting method to the complete set of songs, a model can be acquired, which correctly recognises all subphrases, phrases and themes in all ten songs. The term '*perfect model*' will be used henceforth for this model, which is useful as a reference point for the performance of the refinement methods (see chapter 8). Moreover, the perfect model has been used to verify that the feed-forward classification algorithm, correctly achieves

100% classification accuracy on the ten songs. This result is achieved under the provision that event occurrences which are subsumed by other occurrences of the same event are removed. This can happen in two cases:

- In *repeating events*, an event is recognised as soon as the minimum number of repetitions is exceeded. According to the generic model of the song, as presented in section 7.3, the real event which should be recognised is the longest one. For example, given a repetition of P_1 subphrases, T_1 should start with the first P_1 and end with the last. In addition to this real T_1 event, the classification algorithm will recognise many other subsumed ones.
- In *disjunctive events*, where one definition subsumes the other, the same situation occurs. In the context-sensitive model, this only happens for SP_{41}, which subsumes SP'_{41}.

The automatic parameter setting can be seen as a fully-supervised learning task, which becomes trivial in virtue of the special labelling scheme and with the tolerance of some over-generalisation. Other interesting properties of the task are:

Feed-forward classification tree: The classification model is a tree, rather than a network.

No overlapping events: One feature of the data acquisition process is that there are no overlapping occurrences of units. This could change, if ambient noise was also coded and/or parallel whale songs were examined. Furthermore, there are no overlapping definitions of higher-order song components and therefore occurrences of these cannot overlap.

No negative examples: Since the definition sequences are all unique and non-overlapping, it is almost impossible to misclassify higher-order events in an input sequence of unit occurrences. The only situation where this can occur is by combining subphrases from consecutive phrases. If for instance, two P_1 phrases occur in sequence, the first SP_{11} could be combined with the second SP_{12} to misrecognise an overly long P_1. This, however, requires a distance range in the definition of P_1, which is too long and is rarely encountered in reality.

7.5.2 Context-free model

With the context-free labelling scheme, the mapping between unit types and higher-order song components becomes many-to-many, i.e., one unit type can be used in more than one subphrases. As a result, the simple model-initialisation method described above would not generate sensible parameter settings. Instead, it would produce an overly general model, with very large parameter ranges. For this reason, the context-free model was generated by direct translation of the corresponding context-sensitive one.

An interesting issue is the classification performance of the perfect context-free model. This is less than 100%, due to the extended overlap between subphrase definitions, which leads to the misrecognition of song components at different levels of the song. Figure 7.10 presents the number of *extra* events, i.e., false positives, that are recognised by the context-free model. One interesting result is that even in the context-sensitive case the misclassification of themes is low. There is only one theme which is misrecognised. This is theme T_2 in song 7, where there is a shift in the second unit of subphrase SP'_{12}. All other misrecognised subphrases and phrases are rejected by the sequential and temporal constraints in the model. This illustrates one of the strengths of the temporal classification network for an event recognition task, namely that it can resolve the uncertainty generated at the low, signal-processing level. The number of false negatives is much smaller: SP_{21}, P_2 and T_2 are not recognised with the correct time stamp in song 7, for reasons explained above, and the time stamp of one SP_{22} is wrong in song 9.

In the refinement tasks, which are described in detail in chapter 8, the training data is separated into the input unit sequence, called *input data*, and the higher-order events which should be recognised, called *feedback data*. This distinction is necessary, because there is no direct correspondence between input and output, as in the common vector-based learning tasks. An interesting question which arises is whether the feedback data can be used for training with the context-free model. The problem with the context-free unit labelling is that some subphrases are subsumed by others, due to the multiple use of unit types. The feedback data consists of the real subphrase, phrase and theme occurrences and does not take the subsumption of the model into account. As a result, a large number of negative examples, one for each misrecognised event in Fig. 7.10, will be caused if this data is used for training. If these events are used as negative examples, the refinement algorithm will retract the parameter ranges of the model, in order to exclude them. The resulting model will be the appropriate one for the

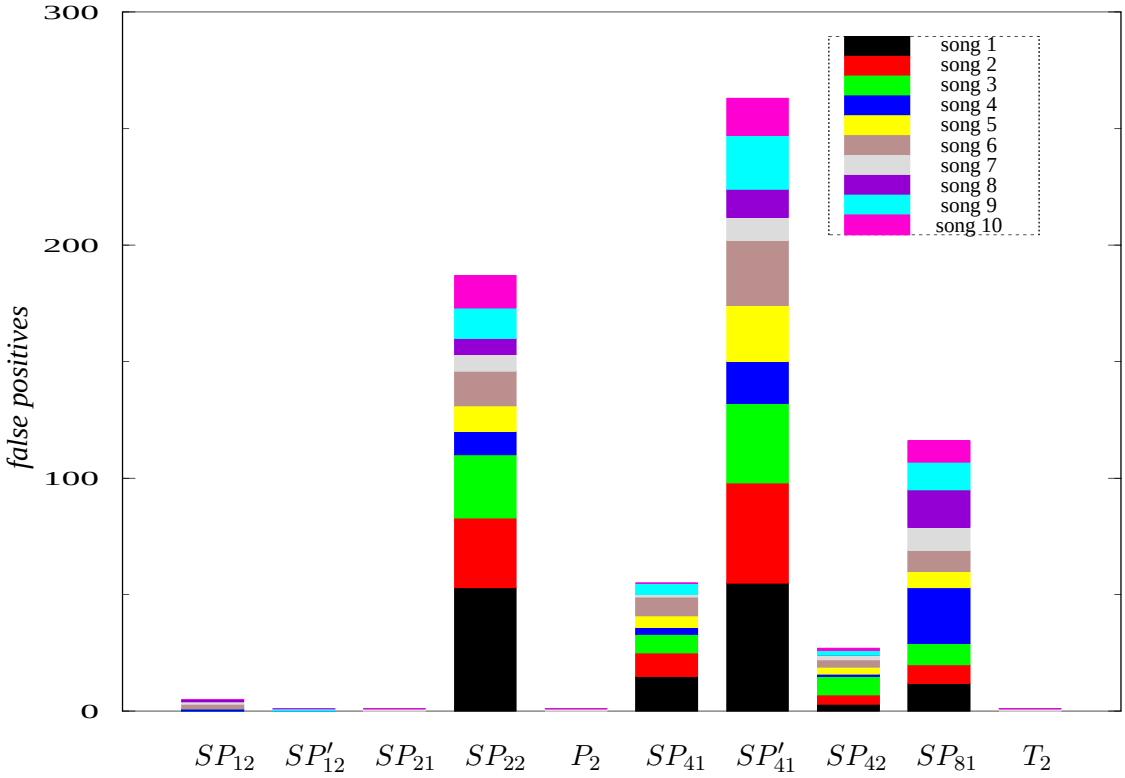


Figure 7.10: Misrecognition of song components by the perfect context-free model.

training data, but it will deviate significantly from the perfect model, which is the basis for the evaluation of the refinement methods. Thus, in order for the perfect model to be of use in the evaluation, the misrecognised events should be provided as positive examples to the refinement algorithm. This can be achieved by using as feedback data the data generated by applying the perfect context-free model on the input data.[7]

7.6 Summary and critique

The song of the humpback whale has a complex and interesting structure. This chapter has examined the construction of a theme-recognition tool for the song that was sung in the North Pacific Ocean in the breeding season of 1977/78. Using spectrograms, a few of the songs from this season have been transcribed and coded as sequences of unit occurrences, which are the basic elements of the song. A TCN model has been presented for the recognition of higher-order components of the song. Finally, the difference between context-sensitive and context-free unit

[7]This only affects the refinement under full supervision, since there are hardly any misrecognised theme occurrences.

classification has been examined and a method was presented for automatically generating the model parameters from a subset of the data.

The approach adopted for the transcription and encoding of the data suffers from several shortcomings:

- The manual transcription process is not a realistic approach and may lead to inaccuracies. It is affected by the knowledge and prejudices of the person performing the transcription and it is prone to human error, especially in the encoded time points. The problem of automating the low-level unit classification process is very important and necessary, before a higher-order, classification model can be of use.
- The context-sensitive unit classification is also unrealistic, unless a system is developed, which can take contextual information into account. Such a system should be able to backtrack and re-evaluate the classification of lower-order events, in the light of the events which have been recognised at higher orders. Although this functionality is not included in the present refinement methods, the TCN framework lends itself to such an extension, since it allows for re-use of lower-order events and recognition of alternative higher-order ones. A similar approach can be taken in the unit classification system, which could provide alternative classifications of the units to the TCN. The TCN could filter them with the use of sequential and temporal constraints. This approach would increase the uncertainty at the unit-classification stage, in a similar way that the use of context-free unit types does.
- The clustering method, which generates the context-free data and model suffers from the problem of not being applied directly to the unit occurrences, but to generic unit descriptions. In other words, each of the context-sensitive units is mapped uniquely onto a context-free unit type. Such a perfect mapping between units and unit types is not realizable with the simple clustering rule used in this chapter. This is a compromise which had to be made, due to the lack of an automatic unit classification system.

An additional issue is the suitability of the problem as a testbed for the methods developed in this thesis. The goal is to apply the classification and refinement methods to the humpback whale song and evaluate their performance. Since these methods were not specifically developed for dealing with this problem, there are several difficulties in achieving a perfect match

between the two:

- One of the main considerations in the TCN representation is the existence of overlapping events in the training data. This aspect of the model is not exploited by the problem presented in this chapter, which examines an isolated event sequence. An interesting extension to the problem would be to look at songs which are sung in parallel by different whales. This happens rather seldom in reality, because singers tend to maintain considerable distance between them. However, this phenomenon has been noticed in a few cases, where due to either the positioning of the recording ship or the properties of underwater sound propagation, the song of one or more whales appeared in the background of the main subject [75]. Additionally, one could artificially mix two songs, either at the level of the recording or even at the level of event symbols, depending on whether the unit classification of mixed songs can be dealt with. This idea has not been pursued further, due to time limitations.
- Some interesting features of the song cannot be modelled with the static structure of the TCN. Examples of this are shifting units and the change of the song over time. Modelling such features of the song requires a more dynamic framework, which models continuous properties of the examined sounds, rather than viewing them as symbolic events.
- Repeating events can be modelled with a TCN, but they present several problems, as mentioned in chapter 5. The result is that they are treated as a disjunction of conjunctive events, which leads to very rigid models. Since repetition is such an important aspect of the song, more work is required to improve the handling of these events by the TCN.

Despite these problems, the task presented in this chapter is challenging and illustrates the applicability of the methods examined in the thesis. There are certainly many open questions, which present opportunities for further work, possibly with the application of completely different methods. The final target of such work would be a system which facilitates the analysis of the humpback whale song and helps the experts discover further features of this very interesting natural phenomenon.

Chapter 8

Experiments on the whale songs

This chapter examines the suitability of the refinement methods proposed in this thesis to a real-world task: the refinement of a model for the humpback whale song. The model is represented by a temporal classification network, as described in chapter 7. The temporal parameters are initialised automatically on a set of songs and then refined, using a separate set of training data. Once refined, the model is evaluated in terms of its classification performance on unseen data and its proximity to the desired (perfect) model. The acquired results show the benefits of model refinement, but also reveal weaknesses of the proposed methods.

8.1 Experimental setup

Chapter 7 examined the structure of the humpback whale song and presented a generic TCN[1] model for the song in one particular season. Ten songs from this season have been transcribed and coded in an event-sequence format, suitable for processing by the classification and refinement methods proposed in the thesis. Each of the ten songs can be seen as a training example for the task of learning to recognise themes, since most themes occur once in each song. In this manner, the songs form a small dataset, which can be used to evaluate the performance of the refinement algorithms.

8.1.1 Four experiments

The evaluation comprises four experiments, using context-sensitive and context-free data under full and partial supervision:

[1] TCN: temporal classification network.

Experiment	Supervision	Units
I	full	context-sensitive
II	full	context-free
III	partial	context-sensitive
IV	partial	context-free

The incremental refinement algorithm (ORA)[2] cannot be part of the experiments, because it has not been extended to deal with repeating events.

In the case of refinement under full supervision (LRA),[3] the input to the refinement algorithm consists of:

- the initial model, which is named here the *expert model*. The temporal parameters of this are set by the model initialisation method, described in section 7.5, chapter 7.
- the sequence of time-stamped units, corresponding to a subset of the ten songs. This part of the training data is named *input data*. Consecutive songs are concatenated into one dataset, by adding an offset to the time stamps of the units in each subsequent song, sufficient to separate the training material.
- the corresponding training feedback, containing all subphrases, phrases and themes which should be recognised. This part of the training data is named *feedback data*. Combined feedback for a set of songs is generated by the same concatenation method as for input data.

The output of the refinement algorithm is the model, with updated temporal parameter settings. The experiments with refinement under partial supervision (RAPS)[4] differ in only one respect: the feedback data contain information only for the themes which should be recognised, not for the phrases and subphrases. Prototypes for the LRA and RAPS algorithms have been implemented in Prolog and are used here in the evaluation of the methods.

8.1.2 Evaluation criteria

The success of the refinement process can be assessed by the following criteria:

1. *Displacement* of the refined model from the perfect model, i.e., the model which correctly classifies subphrases, phrases and themes in all ten songs.

[2]ORA: Optimal Refinement Algorithm.
[3]LRA: Lazy Refinement Algorithm.
[4]RAPS: Refinement Algorithm under Partial Supervision.

2. Classification performance of the model on unseen data.

The first criterion measures the difference between the temporal parameters of the refined model and the parameters of the prefect model. If S_r and D_r are the sets of distance and duration ranges for the refined model and S_p, D_p, the corresponding sets for the perfect model, the displacement of one model to the other is calculated by the sum of the individual displacement values:

$$\delta(S_r, D_r, S_p, D_p) = \sum_{\substack{[s_r^- ..s_r^+] \in S_r \\ [s_p^- ..s_p^+] \in S_p}} (|s_r^- - s_p^-| + |s_r^+ - s_p^+|) + \sum_{\substack{[d_r^- ..d_r^+] \in D_r \\ [d_p^- ..d_p^+] \in D_p}} (|d_r^- - d_p^-| + |d_r^+ - d_p^+|),$$

where $[s_r^- ..s_r^+], [s_p^- ..s_p^+], [d_r^- ..d_r^+], [d_p^- ..d_p^+]$ are individual distance and duration ranges. The lower the values for δ, the closer the refined model is to the perfect one. In the same way, the displacement of the expert model from the perfect model can be calculated. This criterion can also be interpreted as a measure of the amount of improvement to the expert model. Thus if (S_e, D_e) the parameters of the expert model and similarly (S_r, D_r), and (S_p, D_p) these of the refined and the perfect model, the *improvement ratio* is defined as:

$$ir = 1 - \frac{\delta(S_r, D_r, S_p, D_p)}{\delta(S_e, D_e, S_p, D_p)}.$$

The second criterion, i.e., classification performance, can be measured by using the refined model to classify unseen data and comparing the recognised events to the ones which should have been recognised. The test data are constructed in the same way as the training data, but they are reserved for the evaluation of the classification output. Performance measures of interest are:

- Ratio of themes recognised, ignoring their time stamp. If N_u is the set of themes in the unseen data[5] and $N_d \subseteq N_u$ the recognised themes, the ratio is defined as:

$$\rho_t = \frac{|N_d|}{|N_u|}.$$

- Ratio of themes recognised with the correct stamp. If O_u is the set of theme occurrences in the unseen data and $O_d \subseteq O_u$ the recognised ones:

$$\rho_{to} = \frac{|O_d|}{|O_u|}.$$

[5]As a simplification, each theme is counted only once.

- Ratio of recognised subphrases and phrases with the correct time stamp. Similar to themes:

$$\rho_{po} = \frac{|Q_d|}{|Q_u|}.$$

- Ratio of false recognised themes. If O_a is the set of all recognised theme occurrences:

$$\rho_{tf} = \frac{|O_a| - |O_d|}{|O_a|}.$$

- Ratio of false recognised subphrases and phrases:

$$\rho_{pf} = \frac{|Q_a| - |Q_d|}{|Q_a|}.$$

These ratios are similar to the performance criteria used in the Receiver-Operating Characteristics (ROC) analysis of dichotomous classification models. The measures used in the construction of ROC curves are:

$$\textit{Sensitivity} = \tfrac{TP}{TP+FN} \text{ and } 1-\textit{Specificity} = \tfrac{FP}{TN+FP},$$

where TP, FP, TN and FN stand for true and false positive and negative classification. The ratios ρ_t, ρ_{to} and ρ_{po} are measures of sensitivity, while ρ_{tf} and ρ_{pf} are the alternatives of $1-$*specificity* for the event recognition problem. The difference is that TP is used in the denominator of ρ_{tf} and ρ_{pf}, instead of TN, because TN corresponds to all the events which have correctly *not* been recognised. In common classification problems TN can be measured but in these experiments it cannot be.[6]

8.1.3 Sampling method

A basic requirement for the evaluation procedure is that it provides a reasonable estimate of the average-case performance of the system. In the experimental setup used here, this requirement is difficult to satisfy, for two reasons:

- Small amount of data.
- Unbalanced distribution of the training material in the ten songs.

The common solution to these problems is the use of an iterative *leave-one-out* test strategy, which requires that each data point is used as a test set and an average over all runs is obtained.

[6] In the special case of $O_a = 0$ or $Q_a = 0$, ρ_{tf} and ρ_{pf} remain undefined.

Here, this is equivalent to *ten-fold cross validation* and the final result is an average of the ten runs. In each run, one of the songs is used for testing, i.e., evaluation of the classification performance on unseen data, and the other nine for training.

However, the separation of the data into a training and a test set is only part of the problem. There is an additional requirement to generate the expert model, which is given as a starting point to the refinement algorithms. Section 7.5, in chapter 7, described how this can be done automatically, using a subset of the data. This is needed, because there is no expertise available on what the initial parameter values should be. As a result, the training set in each run of the ten-fold cross validation needs to be further subdivided into a subset of songs for initialisation, the *expert set*, and another for refinement, the *refinement set*. The size, as well as the composition of the expert set can have an important effect on the performance of the refinement algorithms. One extreme is to use one song for the initialisation and eight for the refinement of the model. In this case, the temporal parameter ranges of the expert model are very narrow, just covering the one-song expert set. The refinement algorithm has the task of expanding these ranges – contrary to its proximity bias – to cover the eight songs in the refinement set. At the other extreme, eight songs can be used in the initialisation, allowing for little refinement to take place, with the use of the remaining song.

In order to gain a good estimate of the performance of the algorithms, one would need to look at all possible sizes, 1 to 8, of the expert set and average over all possible compositions of it at each size. The first of the two requirements, i.e., varying the number of songs in the expert set, is equivalent to varying the size of the training set in a typical learning problem and is adopted here to construct a learning curve. Thus, $8 \times 10 = 80$ runs are performed for each of the four experiments, where 8 is the number of different sizes of the expert set and 10 the number of runs under ten-fold cross validation. The second requirement however, i.e., examining all possible compositions of the expert set, causes an exponential increase in the number of runs to be performed. For an expert set of size n, there are $\binom{9}{n}$ possible configurations of the set. Thus for all 8 set sizes, the number of runs is:

$$10 \times \sum_{n=1}^{8} \binom{9}{n} = 5,100.$$

This number of runs is not achievable by the prototype implementations used here and therefore it was decided that this dimension would not be explored. Instead, for each of the 80 runs, a

random configuration of the expert set has been generated and the remaining songs were used for training.

8.1.4 Standard of comparison

As a means of comparison, in addition to the refined model, two more models are evaluated:

- The expert model.
- The *goal* model.

The goal model is the equivalent of the perfect model for the training set, i.e., it just covers all nine songs in the expert and refinement sets. The expert model is undergeneralised and the refinement algorithm is required to generalise it, i.e., expand the given ranges. Due to the proximity bias, the generalisation should lead to a model which just covers all nine songs. Therefore, the goal model is an approximation of the desired model after refinement. The perfect model is more general, since it covers all ten songs. This coverage is not achievable by refinement in the experimental setup used here.

Being invariant throughout the 80 runs in each experiment, the perfect model provides a fixed point of reference for the performance of the refinement algorithm. The most important result, however, is the performance of the refined model relative to that of the goal model. Thus, if $S_r, D_r, S_g, D_g, S_p, D_p$ are the sets of distance and duration ranges for the refined, goal and perfect model respectively, the quantities which should be compared are the displacement of the refined model from the perfect model, $\delta(S_r, D_r, S_p, D_p)$, and displacement of the goal model from the perfect model, $\delta(S_g, D_g, S_p, D_p)$. Note also that if S_e, D_e are the parameter ranges for the expert model, the following relation should hold in general:

$$0 \leq \delta(S_g, D_g, S_p, D_p) \leq \delta(S_r, D_r, S_p, D_p) \leq \delta(S_e, D_e, S_p, D_p), \tag{8.1}$$

due to the fact that the refined model is expected to generalise the expert model, but not beyond the goal model. A similar behaviour is expected for the classification performance measures.

The goal model is acquired by applying a variant of the automatic model initialisation algorithm to the refinement set. This algorithm generalises the expert model, i.e., expands the temporal ranges, using the refinement set. It differs from the initialisation algorithm in that it leaves the definitions of events which are missing from the refinement set unchanged, while the initialisation algorithm replaces them with the corresponding parameters of their surrogate

definition. The reason for this modification is that the refinement algorithm handles missing events in the same way and the purpose of the goal model is to act as a standard of comparison for the refined model. It should be stressed, that the goal model is just an approximation of the ideal refined model and it therefore fails in several ways. Even the intuitive relation (8.1) may not hold in some cases. The approximation errors are explained in the context of the results presented in the remaining sections of this chapter.

The goal model for the context-free unit types is constructed by direct translation of the equivalent context-sensitive one. Some interesting results with the overly general "goal" model, acquired by automatic initialisation from the context-free data, as mentioned in section 7.5, chapter 7, are presented in the following sections. The term *pseudogoal model* is used for this overly general model.

8.1.5 Parameter settings for refinement

Another important aspect of the experimental setup involves the setting of the parameters for the refinement algorithms. These parameters are the following:

Positive-to-negative ratio (β). The aim of the refinement is generalisation of the expert model, with the use of positive examples from the data in the refinement set. For this reason, negative examples are ignored by setting $\beta = 1.0$. Under full supervision, there are not many negative examples (see section 7.5, chapter 7) and the value of β should not play an important role. However, under partial supervision, where there is uncertainty about the recognised subphrases and phrases, a large number of alternative events are generated, most of which are negative examples. These examples should be ignored.

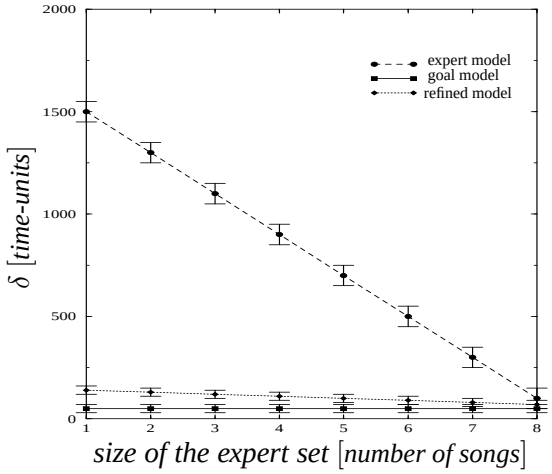
Purity-to-proximity ratio (γ). In the full supervision experiments there is no noise in the data and γ is set to a high value, $\gamma = 0.9$, giving a high data bias. For partial supervision there is increasing uncertainty in the propagation of feedback and the model bias can play an important role in avoiding overgeneralisation. Therefore it was decided to give an equal weight to the two biases, $\gamma = 0.5$. A reasonable alternative would be to vary γ according to the size of the expert set. This approach was not adopted, however, because the relation between γ and the size of the expert set is not known.

Distance and duration windows. In order to avoid biasing the search towards the correct parameter settings, these are set to uniform default levels. All distance windows are set to a minimum of 1, since there are no overlapping events, and a maximum which covers the largest encountered distance. For duration windows, a distinction is made between units and other events. The former tend to be considerably shorter than the latter and if the windows were set to be too large, there would be no discrimination between the proximity of different candidate solutions, due to the disproportionately high denominator of the proximity function (see section 5.5, chapter 5). Therefore, the maximum duration for units in subphrase definitions is set to be an order of magnitude smaller than for other events, still covering all unit durations. The minimum duration is also set to 1. The chosen approach of uniform window settings is problematic, because of the effect it has on the calculation of proximity. An alternative approach would be to use the automatic window determination, by proportional extension of the original windows (see section 5.5). If this were to be chosen, however, it should be made dependent on the size of the expert set, otherwise it could inhibit the required range expansion. This observation suggests the association of the automatic determination of parameter windows with the γ parameter. This idea has not been explored further, due to time limitations.

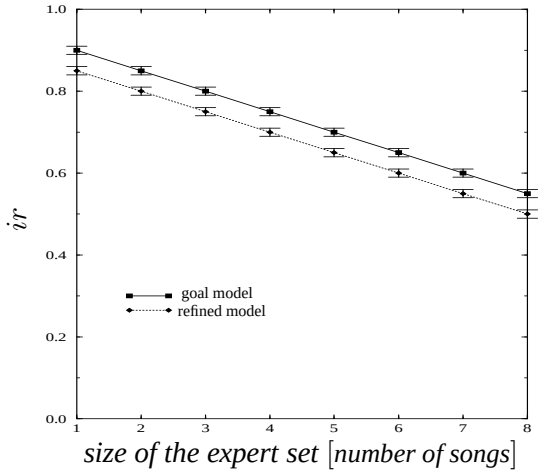
Thus, the parameters used by the refinement algorithms were fixed throughout each of the four experiments, taking values as described above. However, in order to gain an idea of the dependence of the algorithms to the parameters, an additional experiment was performed, in which the performance of the algorithms for different parameter settings was evaluated. The results of this experiment are presented in section 8.4.

8.1.6 Analysis of results

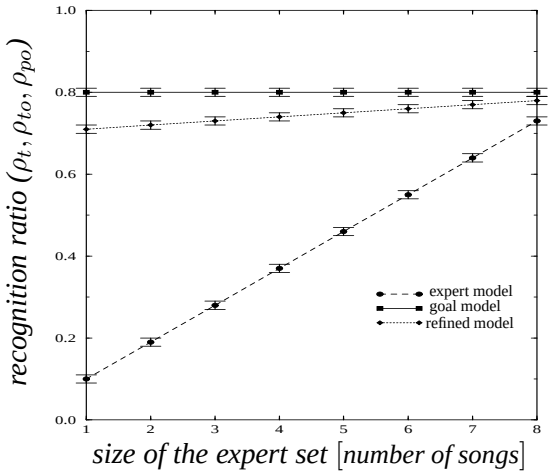
As mentioned in section 8.1.3 above, an 8×10 setup is used for each of the four experiments, averaging over 10 runs for each of the 8 possible sizes for the refinement set. The results of the experiments are analysed with the use of graphs, which portray the behaviour of different models, as the size of the refinement set changes. Figure 8.1 sketches the typical format of these graphs.



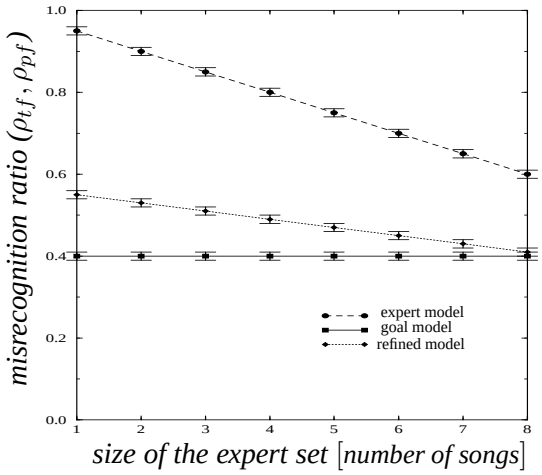
(a) Displacement from the perfect model.



(b) Improvement ratio, over the expert model.



(c) Recognition ratio.



(d) Misrecognition ratio.

Figure 8.1: Idealised curves for the analysis of the results.

On the horizontal axis of each graph, the size of the expert set varies from 1 to 8. The vertical axes represent the various criteria that were introduced in section 8.1.2, i.e., the displacement and classification performance measures. Each curve in a graph illustrates the behaviour of one of the three models: expert, refined and goal. The basis for comparison is always the perfect model. The construction of a curve is based on eight points, each of which is an average over ten runs, i.e., the result of 10-fold cross validation. The error bar shows the standard error at every point.

As the size of the expert set increases, the expert model approaches the perfect model, while the goal model remains largely unaffected, as shown in Fig. 8.1(a). The displacement of the refined model from the perfect also decreases, due to the reduced displacement of the expert from the perfect model. At the same time, the size of the refinement set reduces causing the displacement of the expert from the refined model to decrease. The three curves should not intersect each other, due to the relation presented in (8.1).

The improvement ratio graph, in Fig. 8.1(b), presents the same information as Fig. 8.1(a), but in a different format. This time the amount of improvement by the refined and goal models on the expert model is measured. A value of $ir = 1.0$ means that the displacement of the corresponding model from the expert model is as large as that of the perfect model, indicating that the refined, or goal, model is very close to the perfect model. As the size of the expert set increases, the displacement of the expert from the perfect model decreases, leaving less room for improvement.

The recognition ratio for themes and phrases, i.e., the proportion of recognised events in the test set, takes the form shown in Fig. 8.1(c). The expert model is expected to miss the recognition of many events, especially when the expert set is small. The refined model should improve this result, approaching the goal model. As the size of the expert set increases, the degree of improvement decreases and the three curves converge. Similarly, the misrecognition ratio is shown in Fig. 8.1(d). The expert model is expected to misrecognise a larger number of events than the other two models and improve as the size of the expert set increases.

The fact that the curves for the refined model lie between the two other curves is a result of (8.1) and will hold to the same extent that this relation does. There are cases where (8.1) was violated, mainly due to approximation errors in the goal model. These are discussed as they are observed in the following sections.

8.2 Refinement under full supervision

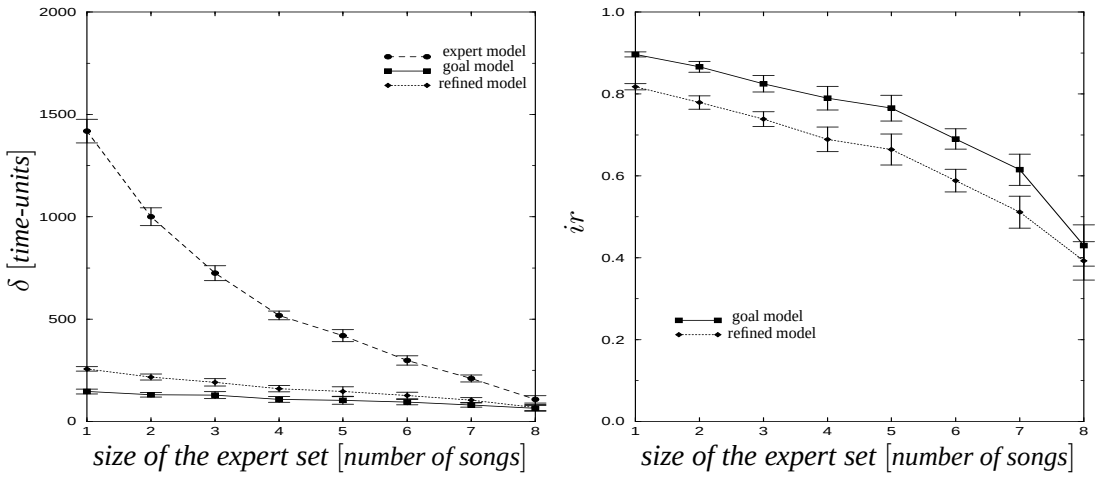
8.2.1 Context-sensitive units (I)

The first experiment examines the simplest case of the refinement task:

- the model and the data use context-sensitive units,
- supervision is provided for all phrases, subphrases and themes in the model.

Under these assumptions, there is a many-to-one mapping between units in the input data and subphrases in the feedback data, i.e., each unit is only used to recognise one subphrase. This mapping should allow the refinement algorithm to extend the expert model so that it exactly covers the refinement data. Negative examples only appear when the distance window does not prohibit the combination of subphrases from consecutive phrases, as discussed in section 7.5, chapter 7. For example, the uniform distance window, used for all events, may be large enough to allow the combination of a SP_{91} occurrence with the SP_{92} of the subsequent P_9 phrase, since both these subphrases do not contain repeating units and are usually short. However, such a negative example can be trivially excluded because the distance between the subphrases is far outside the range required for covering the positive examples.

Given the simplicity of the task, the refined model is expected to be almost identical to the goal. Figure 8.2(a) presents the δ values for the three examined models and Fig. 8.2(b) shows the improvement ratio for the refined and the goal models.



(a) Displacement from the perfect model.

(b) Improvement ratio.

Figure 8.2: Displacement results, full supervision with context-sensitive units.

The δ values of the refined model lie between those of the goal and the expert model, but they do not coincide with the goal. The reason for this is that the parameter ranges for the goal are further expanded than these of the refined model and are in fact larger than what is needed for covering the nine songs. This is a result of the overgeneralisation, which was discussed in

section 7.5, chapter 7, caused by exceptional multiple usage of units in subphrase definitions. Examples of such units are g, which is used in SP_{41} and SP'_{41}, and x, used twice in SP_{91}. The refined model provides a more accurate cover of the refinement set, since it uses the feedback information to determine the mapping between unit and subphrase occurrences. However, it has a larger displacement from the perfect model than the goal model, since the latter two are generated in the same way. The effect is smaller as the size of the expert set increases, because the expert model, which incorporates the same kind of inaccuracy, moves closer to the goal.

The displacement of the goal model itself from the perfect model is decreasing as the size of the expert set increases. This is due to the modification of the initialisation algorithm, used for the construction of the goal model. The initialisation algorithm presented in section 7.5 uses surrogate event definitions to set the parameters for missing events. For instance, subphrase SP_{12} acts as a surrogate for SP'_{12}, which occurs in only one song. Thus, if SP'_{12} does not appear in the set of songs used for initialisation, the parameters in its definition are copied from the definition of SP_{12}. The modified version of the initialisation algorithm, used to generate the goal model, does not change the definition of events which are missing from the refinement set. As the size of the expert set increases, more songs move from the refinement to the expert set, increasing the amount of overgeneralisation in the expert model, since the surrogate ranges become larger. In other words, the ranges in SP_{12} will be larger when the expert set consists of seven, instead of two songs and so will be the ranges in SP'_{12}, if it is missing from the expert set. The correction of the problem, in the construction of the goal model, will thus have less effect as the size of the refinement set decreases, leading to a more general goal model, closer to the perfect one.

Despite the fact that it essentially depicts the same information as Fig. 8.2(a), the improvement ratio graph in Fig. 8.2(b) provides further insight into the performance of the refined model:

- As expected, the improvement ratio is decreasing as the size of the expert set increases.
- There is still a substantial improvement to the expert model even when there is just one song used for refinement, i.e., the refinement set is of equal size to the unseen test set.
- Less predictably, the difference between the improvement ratios of the refined and

the goal models is almost constant. This shows that the difference between the displacement of the refined model from the perfect model and the displacement of the goal model from the perfect model varies in the same way as the displacement of the expert model from the perfect model, i.e., at two different expert set sizes i, j:

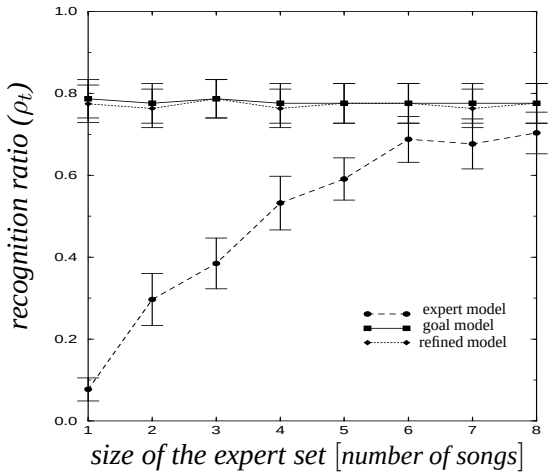
$$\frac{\delta_i(S_r, D_r, S_p, D_p) - \delta_i(S_g, D_g, S_p, D_p)}{\delta_i(S_e, D_e, S_p, D_p)} \approx \frac{\delta_j(S_r, D_r, S_p, D_p) - \delta_j(S_g, D_g, S_p, D_p)}{\delta_j(S_e, D_e, S_p, D_p)}$$

The reason for this is that the convergence of the refined model and the goal model curves in Fig. 8.2(a) are caused by approximation errors in the expert model.

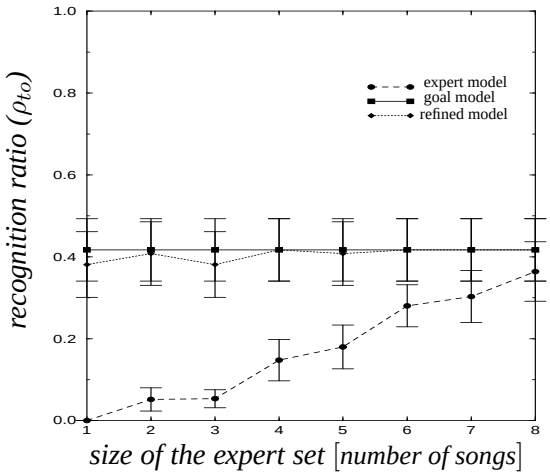
In terms of classification performance, the goal and the refined models are virtually indistinguishable. 100% accuracy on the training set is achieved by both of them in all measures, i.e., there are no false positives or negatives for the refinement and expert sets. Figures 8.3 and 8.4 summarise the performance on the test song. Figure 8.3(a) shows the performance of the three models in recognising themes, disregarding the theme's time stamp and the fact that themes may occur more than once in the song. Both the goal and the refined model recognise roughly 80% of the themes that should be recognised in the test song. The average number of themes per song is 8, out of which the two models recognise roughly 6.3, on the average. The themes which are not recognised are usually the ones which do not contain a large number of phrase repetitions, e.g. the transition themes T_{56} and T_{24}, or theme T_6, which consist of a single phrase. The reason for this is that the recognition of the themes is linked to the correct recognition of one or two phrases, which may be prevented by insufficient generalisation.

The picture is different for the recognition of themes with their correct time stamp, Fig. 8.3(b). The performance of the goal and the refined model are still very similar, but they both have a low recognition ratio. Only about 40%, of the themes in each song are recognised with the correct stamp. The reason for the low recognition ratio at the theme level is the accumulation of error from lower levels. In other words, a theme will not be recognised, unless all of the lower-order events which support it are correctly recognised. If one subphrase is not recognised with the correct time stamp, the corresponding theme will not be recognised either. The goal and the refined models are not general enough to cover all ten songs and recognise correctly all events in the test song. However, they do recognise most of the events correctly. Figure 8.3(c) shows that about 90% of phrases and subphrases are correctly recognised by the goal and the refined models; the latter having a slightly lower recognition ratio, due to its tighter coverage of the refinement set. The 10% error at the level of subphrases and phrases is

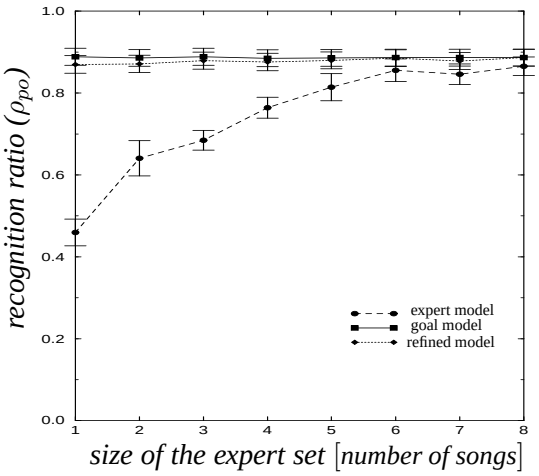
sufficient, though, to cause the larger error at the theme level.



(a) Themes (without time stamps).



(b) Theme occurrences (with time stamps).

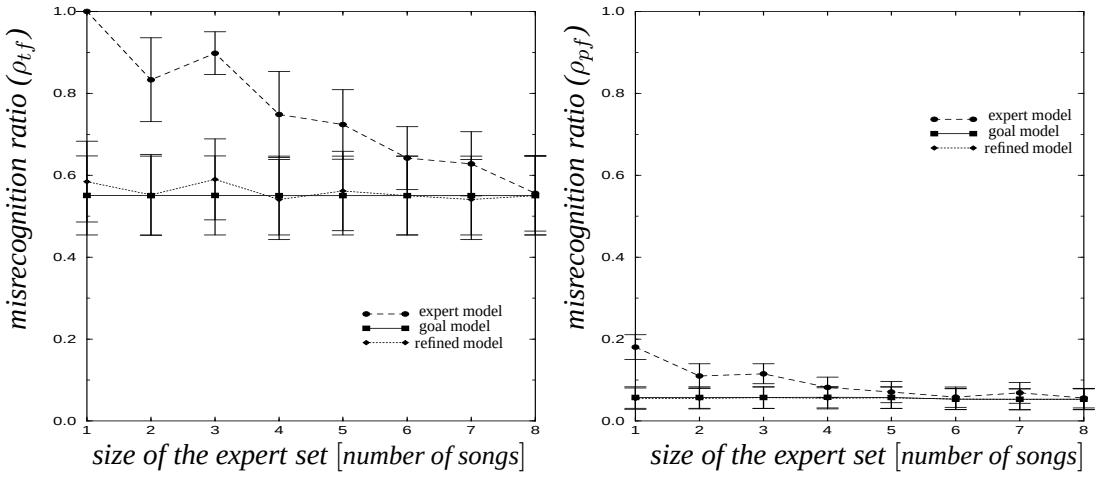


(c) Phrase/subphrase occurrences (with time stamps).

Figure 8.3: Recognition ratio, full supervision with context-sensitive units.

Similar results are acquired for the misrecognition ratios shown in Fig. 8.4. The expert model starts with a theme misrecognition ratio of 1, Fig. 8.4(a), when the expert set is very small, meaning that even the small number of themes that are recognised are not correctly stamped, and the situation improves with the size of the expert set. For lower-order events, Fig. 8.4(b), the performance of all models is better, the goal and the refined model having only

about 5% of the recognised events incorrectly stamped.



(a) Theme occurrences (with time stamps).

(b) Phrase/subphrase occurrences (with time stamps).

Figure 8.4: Misrecognition ratios, full supervision with context-sensitive units.

One cause for concern in the above results is the size of the standard error in the performance estimates, particularly in the theme recognition and misrecognition ratios. This is caused by the small number of themes involved. For example, the misrecognition of one theme in song 4 causes a change of 14 percentage points in performance. However, even when the error bars overlap, the order of the three models is usually the same. In other words their performance varies in the same way for different test songs. This is always true between the expert model and the other two. Between the goal and the refined model there are some exceptions, when the former outperforms the latter. An example of this is the misrecognition ratio in Fig. 8.4(a), where the overly general goal model can get a worse result than the more specific refined model.

In summary, the refinement algorithm has achieved the expected performance in this experiment, improving the expert model to reach the goal. According to all criteria that were set, the goal performance has been reached, irrespectively of the size of the expert set, i.e., the amount of information included in the expert model. These results serve as a basis for comparison with the more complex tasks.

8.2.2 Context-free units (II)

With the use of context-free unit types the many-to-one mapping of units to subphrases, is replaced by a many-to-many one, where units are reused in several theme definitions. The main effect of this on the refinement task is an increase in the number of negative examples generated for subphrases. For example, a sequence of three units B, C, B will generate an example for SP_{41}, even if in reality it is a subsequence of SP_{51}. However, the number of negative examples is reduced by the generation of artificial feedback data. As mentioned in section 7.4, the training feedback is generated by the perfect context-free model, in order to avoid non-excludable negative examples. Thus, in the example above, both the SP_{41} and the SP_{51} subphrases will be included in the feedback data, generating two positive examples. The reason for this is that the perfect model cannot distinguish between the two subphrases. The remaining negative examples are easily excludable and will be ignored, because $\beta = 1.0$.

As a result of the above simplification of the task, the performance of the refinement algorithm is similar to that in the first experiment. Similar to Fig. 8.2(a), Fig. 8.5 shows the displacement of the three generated models from the perfect one. The results are almost identical to those in Fig. 8.2(a). The only difference between the two figures is a small shift in each of the curves, which is due to the structural differences of the two models and approximation errors in the translation of the context-sensitive models to context-free ones. An example of the former situation is the merging of the context-sensitive e and e' units into a common unit type D. The translation of context-sensitive to context-free definitions, reduces the size of the expert models and therefore their displacement from the perfect model. On the other hand, there are inaccuracies in the translation, which have the opposite effect. An example of this is the translation of the definition for theme T_7. In the context-free model, this theme is merged with T_6, to give $T_{6/7}$. However, the phrase P_6 is not explicitly modelled, because the theme consists always of a single phrase, and so there is no information about the duration of P_6 to be used for the definition of $T_{6/7}$. As an approximation, the duration of $P_{6/7}$ is set solely by T_7 and does not necessarily cover T_6. This type of error, increases the displacement of the translated models, expert and goal, from the perfect model. The two types of error counteract to cause small shifts in the three curves of Fig. 8.5.

The fourth curve in Fig. 8.5 corresponds to the pseudogoal model, which is generated automatically from the context-free data. As mentioned in section 8.1.4, this model is overly general

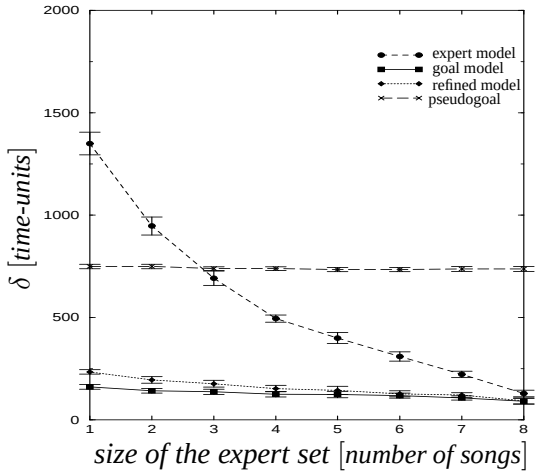
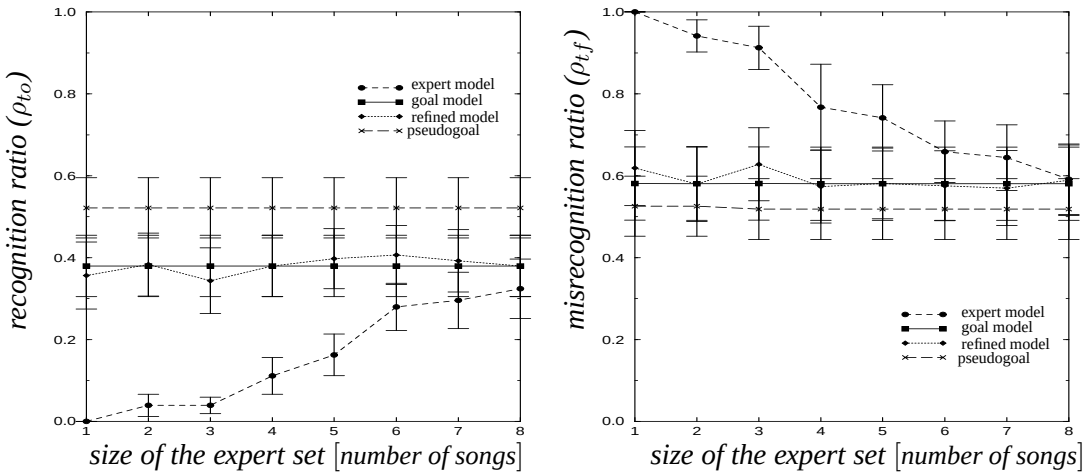


Figure 8.5: Displacement results, full supervision with context-free units. The fourth curve corresponds to the overly general pseudogoal model.

and has a large displacement from the perfect model. Interestingly, however, the classification performance of this overgeneralised model is better than that of the others. This is shown in Fig. 8.6, where the overgeneralised model recognises in average about one theme more than the goal and the refined model do and at the same time has fewer theme misrecognitions. This is a result of the fact that only the lower levels of the TCN are overly general. At the higher levels there are not many differences between the context-free and context-sensitive models and the automatic initialisation generates similar parameter settings. In the classification stage, the expected effect of overgeneralisation is the misrecognition of events. At the theme level, the temporal constraints are set correctly, preventing misrecognition. In fact, the low misrecognition ratio of the overgeneralised model, shows that this is preferable to an undergeneralised model. The goal and the refined model fail to cover some of the repeated phrases in the themes of the test song, resulting in either incorrect time stamps or failure to recognise the theme, if it consists of only one or two phrase occurrences. On the other hand, the overgeneralised model, recognises more of the occurring phrases, some of which are rejected by the theme definitions. This argument is verified by the high misrecognition ratio of the overly general model at the subphrase and phrase level, Fig. 8.7.

Another unexpected phenomenon, appearing in Fig. 8.6(a) is the higher theme recognition ratio of the refined model from the goal for some expert set sizes. This is due to the translation errors mentioned above. For example, the refined model contains the correct parameter settings

for the definition of $T_{6/7}$, i.e., those needed to cover the refinement set, while the goal is undergeneralised. For this reason the recognition performance of the refined model can be higher than that of the goal model.



(a) Theme occurrence recognition.

(b) Theme occurrence misrecognition.

Figure 8.6: Theme recognition and misrecognition ratios, full supervision with context-free units.

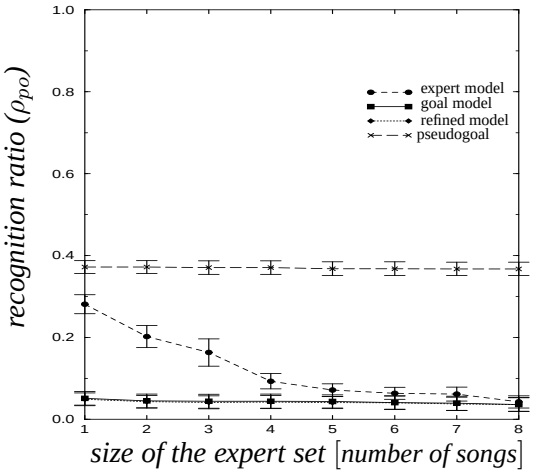


Figure 8.7: Phrase/subphrase misrecognition ratios, full supervision with context-free units.

8.3 Refinement under partial supervision

Under partial supervision, the refinement task becomes more complex. The feedback data consist of only the theme occurrences that are required to be recognised and all other events are treated as hidden. The refinement algorithm constructs all of the examples which could potentially be of interest and allocates to phrases and subphrases the feedback that it receives for themes, weighting the examples according to their model-based and data-based recognition beliefs. The number of examples generated by this process is much larger than under full supervision and the feedback assignment algorithm is responsible for labelling most of them as negative. The large number of resulting negative examples does not affect the search for new parameter settings, because negatives are ignored by setting $\beta = 1.0$.

The extensive use of repeating events in the model of the song causes a technical problem under partial supervision, namely a combinatorial explosion in the number of generated examples. This is especially true for the themes T_1 and T_2, which combine the following features:

- The themes consist of large numbers of phrase repetitions.
- One of their subphrases is also a long sequence of repeated units.

Despite the simplifying assumptions in the handling of repeating events, presented in section 5.6.2, the above combination of repeating events causes substantial problems. A number of subsumed subphrase examples are generated, which lead to an equally large number of phrases and to a much larger number of themes. The result is an exponential increase in the size of the relative event support trees (RESTs). In the condensed REST format, which is used in the forward construction phase, the effect is not very large. However, the refinement algorithm requires the RESTs to be flattened and in some cases the memory requirements for this translation are prohibitively high[7]. The problem is worse as the size of the refinement set increases, i.e., for small expert sets, but can be solved by replacing large songs in the refinement set, e.g. song 1, with smaller ones, which the random selection process has allocated to the expert set. This is done in roughly equal proportion for context-sensitive and context-free units and is discussed in more detail in the following sections.

[7]There are cases where the flat REST consists of more than 10^8 nodes.

8.3.1 Context-sensitive units (III)

Refinement of the context-sensitive model is the easier of the two partial supervision tasks, since for a particular sequence of units there is usually just one subphrase that can be recognised. The main problem here is caused by repeating events, which, as mentioned above, cause the construction of many negative examples. Further to the problem of handling the large number of examples, this can sometimes lead to unavoidable errors in the refinement. As an example, assume the following sequence of recognised a units and simplify the definition of SP_{11} to require 1 to 12, instead of 4 to 12 repetitions (see Fig. 7.3(a), chapter 7) of a:

$$a(0,2), a(3,5), a(7,8), b(9,15), c(17,20), d(22,30),$$

which should lead to the recognition of $SP_{11}(0,8)$, $SP_{12}(9,30)$ and $P_1(0,30)$. Assume also that initial parameter ranges are defined as follows:

`duration`$(SP_{11}, a, [2..4])$,
`duration`$(SP_{12}, b, [5..8])$, `duration`$(SP_{12}, c, [3..6])$, `duration`$(SP_{12}, d, [5..8])$,
`distance`$(SP_{11}, a, a, [1..4])$, `distance`$(SP_{12}, b, c, [2..8])$, `distance`$(SP_{12}, c, d, [1..5])$,
`duration`$(P_1, SP_{11}, [4..20])$, `duration`$(P_1, SP_{12}, [12..30])$,
`distance`$(P_1, SP_{11}, SP_{12}, [1..5])$,

where the only required change is to expand the duration constraint on a from $[2..4]$ to $[1..4]$. In the forward construction of example sequences, the above sequence of a units will lead to the following examples for SP_{11}:[8]

$$\begin{aligned} SP_{11}(0,2): &\quad a(0,2) \\ SP_{11}(0,5): &\quad a(0,2), a(3,5) \\ SP_{11}(0,8): &\quad a(0,2), a(3,5), a(7,8) \\ SP_{11}(7,8): &\quad a(7,8) \end{aligned}$$

Suppose now that $P_1(0,30)$ is correctly labelled positive by the feedback allocation algorithm. The problem is which of the four SP_{11} subphrases will be chosen to support $P_1(0,30)$. The desired answer is the third sequence. However, the algorithm will choose the second, as it does not require any changes to the initial parameters. This type of problem accounts for many of the errors that were made by the refinement algorithm in the experiment.

[8] Subsumed subsequences which satisfy the initial ranges are not generated, (see section 5.6.2, chapter 5).

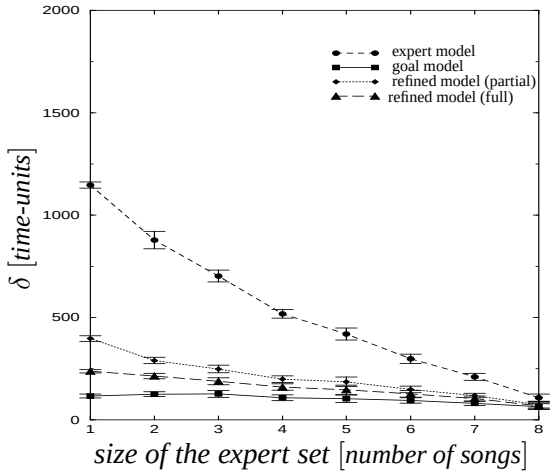


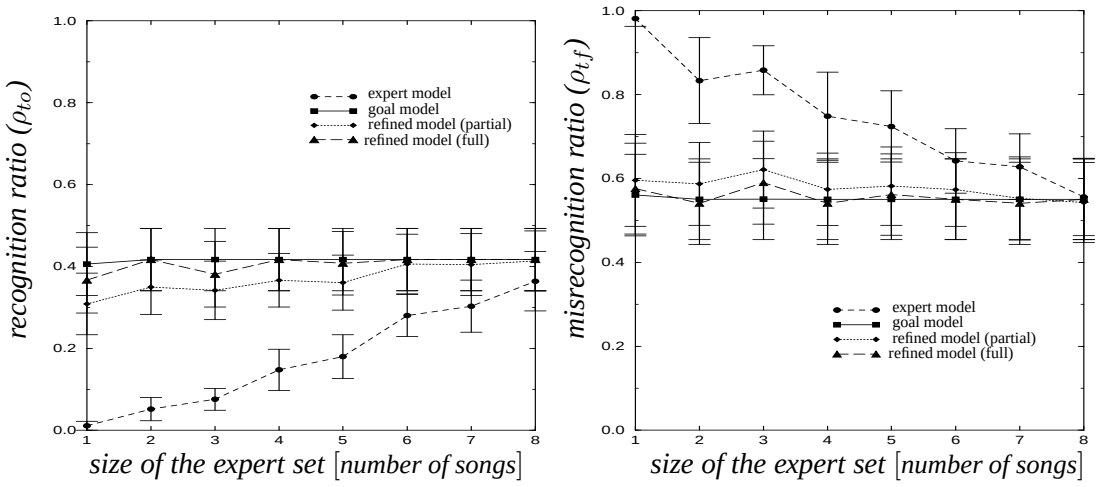
Figure 8.8: Displacement results, partial supervision with context-sensitive units. The fourth curve corresponds to the refined model under full supervision and is included to facilitate the comparison of the tasks.

Figure 8.8 presents the results for the three models, in terms of displacement from the perfect model. In addition, the displacement of the refined model under full supervision is included for comparison. This curve is not identical to that in Fig. 8.2(a), as some of the random splits between expert and refinement set have been rearranged. There is one such rearrangement when the expert set size is 3, three when it is 2 and nine when it is 1. In almost all cases the rearrangement was done by replacing song 1, which is the largest, in the refinement set with a smaller song from the expert set.[9] This rearrangement of the experimental setup has the following effects:

- The expert and goal models are further generalised, at the smaller expert set sizes and therefore their displacement from the perfect model is reduced.
- There is very little variance in the performance of the expert model for expert set size of 1, as the same song, song 1 is used to build the expert set in nine out of ten runs.
- The refined model, under partial supervision seems to be adversely affected by the rearrangement. This is not the case under full supervision, where there is a small improvement. This improvement is due to the more general expert model (see section 8.2.1). The detrimental effect under partial supervision is not easily explainable by a single phenomenon.

[9]There is just one exception to this where song 2 was swapped, because song 1 was used as the test set.

Overall, the performance of the refinement algorithm is not much worse than under full supervision, despite the substantial reduction in feedback information. This is also verified by the results according to the other criteria. Figure 8.9 presents the theme recognition and misrecognition ratios, which draw a similar picture as the displacement results. The results acquired for subphrases and phrases are similar and for this reason they are omitted. The lower misrecognition ratio of the refined model under partial supervision, for small expert sets, indicates that the rearrangement of the experimental configuration causes undergeneralisation of the model, which agrees with the displacement results in Fig. 8.8.



(a) Theme occurrence recognition.

(b) Theme occurrence misrecognition.

Figure 8.9: Theme recognition and misrecognition ratios, partial supervision with context-sensitive units.

8.3.2 Context-free units (IV)

The use of context-free unit types under partial supervision complicates the task substantially. Due to the extensive sharing of unit types between subphrases, there are many more possibilities of generating event examples and the task of the refinement algorithm is to select the true positive ones. The feedback data for this task is identical to those in experiment III, i.e., correctly stamped theme occurrences. Hence there is no additional information provided by artificial data. One effect of the additional complexity of the task is that the memory requirements for the flat RESTs increases. As a result two more experimental configurations needed

to be rearranged, in order to have ten runs at each examined size of the expert set. One at the expert set size of 3 and one at 2.

The results acquired in this experiment are very encouraging. The performance of the algorithm, according to all the criteria used, is almost identical to that using context-sensitive units. Figures 8.10 and 8.11 show the performance of the algorithm in terms of the displacement of the refined model from the perfect model and the theme recognition and misrecognition ratios. The results for phrases and subphrases are similar. Thus, the increased uncertainty in the definition of the context-free model does not seem to affect the performance of the refinement algorithm.

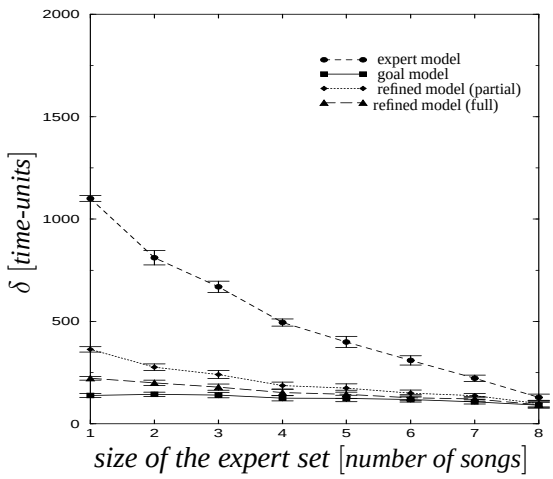
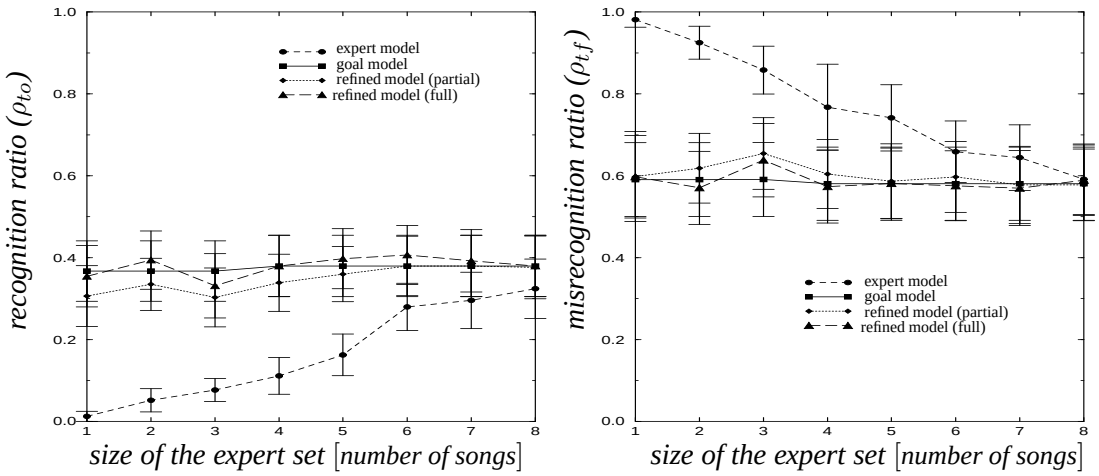


Figure 8.10: Displacement results, partial supervision with context-free units. The fourth curve corresponds to the refined model under full supervision.

8.4 Varying the parameters for refinement

The refinement algorithm uses two parameters, β and γ, which were set for the four experiments described above to problem-specific values. The first, β, which is the relative weight between positive and negative examples, was set to ignore negative examples completely, i.e., $\beta = 1.0$. The second, γ, which provides the relative strength between the model and the data, was set at different values for full and partial supervision. For full supervision it provided a



(a) Theme occurrence recognition.

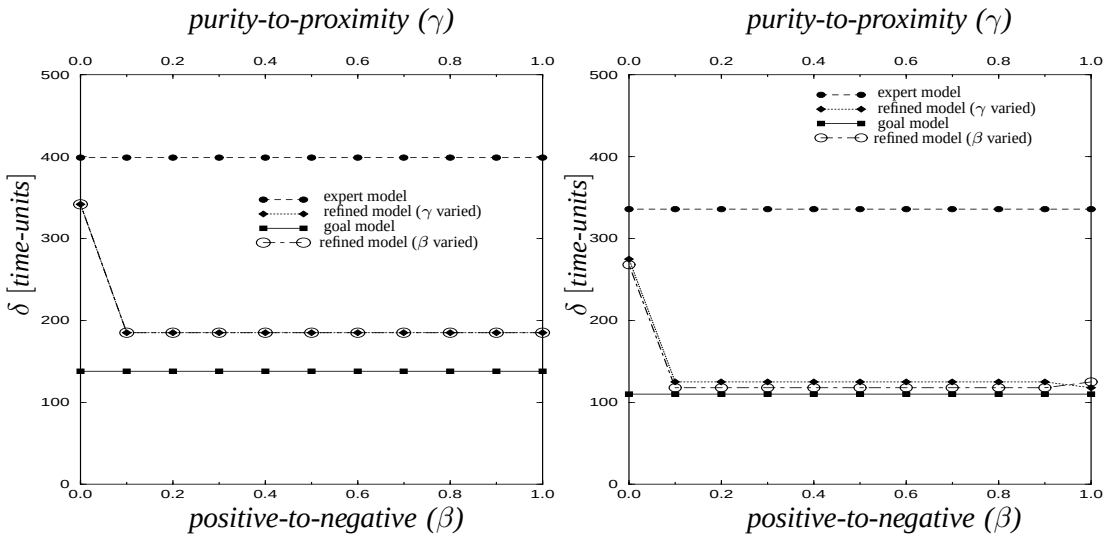
(b) Theme occurrence misrecognition.

Figure 8.11: Theme recognition and misrecognition ratios, partial supervision with context-free units.

bias for data, $\gamma = 0.9$, while under partial supervision data and model information was combined in a unbiased way, i.e., $\gamma = 0.5$. This section examines the dependence of the acquired results on the choice of β and γ values. The investigation is limited and is only intended to give an indication of the effect of the two parameters, rather than be an exhaustive analysis of the behaviour of the algorithm.

For this experiment, only one configuration of songs is used, i.e., the songs in the expert, refinement and test set are fixed throughout the experiment. Each of the two parameters is varied in turn, while the other remains fixed at the value assigned to it in the corresponding experiment. This process is repeated for each of the four experiments described in the previous sections. The experimental configuration which was used consists of six songs in the expert set and three in the refinement set. This size of refinement set was chosen on the basis of computational cost and room for improvement, i.e., large enough displacement of the expert model from the perfect. Out of the ten configurations used in the ten-fold cross validation for an expert set size of 6, the one which achieved an improvement ratio closest to the mean was chosen. The reason for using the displacement measure in the selection of the configuration was that its standard error for the ten runs was smaller than that of other measures. For the same reason, the displacement criterion is used in the rest of the investigation in this section.

Figure 8.12 presents the acquired results under full supervision. Each graph combines the results for both parameters and includes for comparison the δ values for the expert and the goal model. The behaviour of the algorithm with context-sensitive and context-free units is almost



(a) Context-sensitive units.

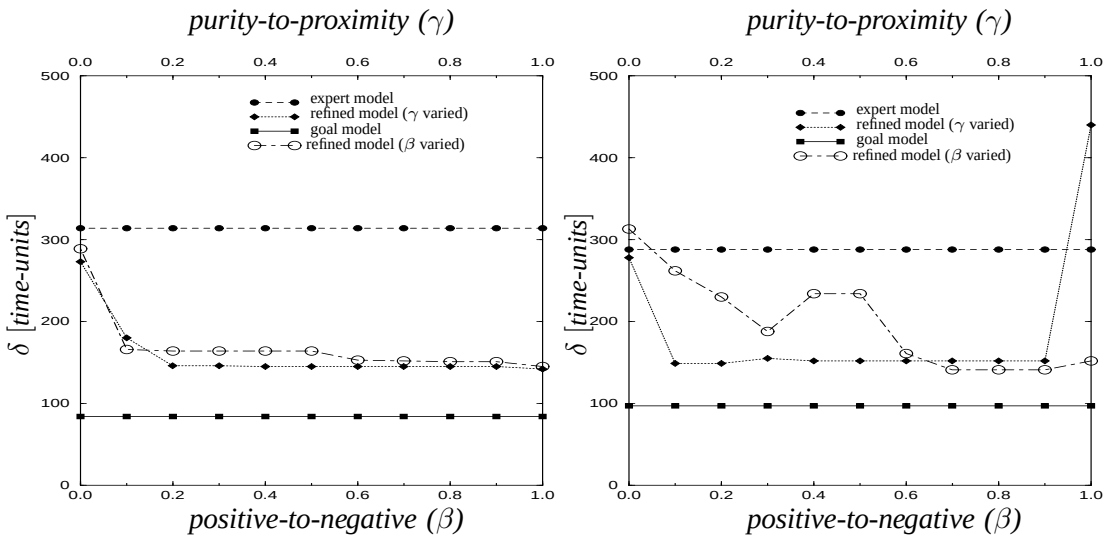
(b) Context-free units.

Figure 8.12: The effect of parameter variations under full supervision.

identical. When varying the β parameter, the performance remains unaffected by the changes, except when $\beta = 0$. At this extreme value of β, positive examples are ignored and since there are no negative examples to cause a change to the model, the expert model remains almost unaltered. There is one exception to this, which causes the small improvement of the expert model, even when $\beta = 0$: if the expert model does not cover at least one positive example of each event in the refinement set, then it is not considered to be a candidate solution. When this happens, the candidate solution closest to the initial ranges, but not equal to them, is selected. The behaviour of the algorithm, when varying γ, is identical for the context-sensitive units and almost identical for context-free unit types. This is mainly due to the effect of large parameter windows, which make the difference between proximity values very small. Unless $\gamma = 0$, purity dominates in the calculation of the overall cost of candidate solutions, even when it is given a low weight. When $\gamma = 0$, i.e., purity is ignored, the minimum amount of change to the expert model is achieved, as described above. The small variation in the performance of

the algorithm, when using context-free unit types, is due to a combination of an approximation error in the expert model and some rare negative examples, which are not excluded by the original ranges.[10]

The situation is slightly different under partial supervision. When context-sensitive units



(a) Context-sensitive units.

(b) Context-free units.

Figure 8.13: The effect of parameter variations under partial supervision.

are used, Fig. 8.13(a), the behaviour of the algorithm is very similar to that under full supervision. The main difference is the existence of more negative examples, which influence the choice of new temporal parameters, causing undergeneralisation of the model for low values of β. As expected, when context-free units are used, Fig. 8.13(b), this phenomenon is more prevalent and in the extreme case of $\beta = 0$, the refined model becomes worse than the expert model. Another interesting phenomenon is the effect of complete disregard for proximity, i.e., $\gamma = 1$: the refinement algorithm attempts to cover all positive examples, ignoring the original parameter settings and overgeneralising the model. The refined model in this case is similar to the overgeneralised pseudogoal model examined in section 8.3 and has a larger displacement from the perfect model than the expert one. The results of this experiment provide an indication

[10] Song 7, which is the only one containing an example of SP'_{12}, participates in the refinement set. The algorithm tries to use this information to correct the initial guess for the parameters of this rare subphrase and its success is affected by the choices of β and γ.

of the difficulty of the task and reinforce the positive results presented for experiment IV, in section 8.3.

The main conclusion from the above investigation is that the selected parameter values in the setup of the four experiments have chosen an area of the parameter space which provides good performance and insensitivity to small parameter variations. This is true even in the complex task of experiment IV, where the sensitivity of the algorithm to the parameter settings increases. In other words, if sensible, problem-specific parameter values are chosen, their exact values are not critical.

8.5 Summary and critique

This chapter has examined the performance of the algorithms, developed in the thesis, on the task of refining models of the whale song. The algorithms have been shown to handle the task at various levels of difficulty. They have improved in all cases the model initially provided to them and brought it close to the estimated upper limit of their performance. Particularly interesting are the good results acquired for the most difficult and realistic variant of the task, i.e., refinement under partial supervision, with context-free units. A further investigation was performed, which gave some indication of the dependence of the performance results on the choice of parameter values for the algorithms. This has shown that the exact values of these parameters are not critical, but it is important that they are assigned sensible values, in the context of the problem that is being tackled.

Despite their good performance, the refinement algorithms have been shown to suffer from one major problem: exponentially increasing memory requirements. The source of the problem is the use of flat RESTs in the search for new settings of the temporal parameters. The problem is particularly serious for repeating events, suggesting that more work is needed to improve the handling of this event type. In a more general context, the format of the memory structure needs to be re-examined and optimised, so that only the necessary parts of a REST are stored, in a more compact way.

In terms of the experimental results presented in this chapter, a desirable improvement is that of the initialisation method, used in generating the expert, goal and perfect models. This problem became apparent in the first two experiments, where the acquired refined models differed from the goal, due to approximation errors in the initialisation process. A more intelligent

algorithm, designed especially for the whale song model, should be able to give more accurate estimates of the desired model for a set of songs. This improvement was not attempted here, as the intention was not to achieve optimal results on the particular problem, but to examine the behaviour of the refinement algorithms.

Another area of possible improvement is in the analysis of the effects of the parameters β and γ. The results presented in section 8.4 are only indicative, due to the limited nature of the investigation undertaken. A more thorough analysis is needed to gain a better insight into the role of the parameters in refinement. An important requirement for such an analysis is the use of improved implementations of the algorithms, rather than the prototypes, which have been used here.

Finally, it should be noted again that the task which was used in the analysis presented here, i.e., the humpback whale song model, does not make full use of the features of the refinement algorithms. In particular it does not contain overlapping events, which was a major criterion in the design of the methods. As mentioned in chapter 7, one could artificially produce such a problem, using the whale songs. However, before attempting such a modification of the problem, it is useful to re-examine the unit classification method and possibly acquire automatically classified data. The reason for this is that the artificial translation of context-sensitive to context-free data, proposed in chapter 7, may introduce uncertainty in the task, which is unrealistic or inappropriate.

Part V

Epilogue

Chapter 9

Conclusions

This thesis has presented a graphical representation for event recognition models and methods for refining the temporal parameters of such models. The proposed methods are novel and specifically tailored to event recognition problems. The applicability of the representation and the performance of the refinement methods have been evaluated using data from a real-world problem, namely the thematic analysis of humpback whale songs. In order to make the task of the thesis manageable, the scope of the work has been restricted by a number of assumptions which are summarised in this chapter. The main issues which were discussed in the thesis are re-examined here, focusing on the merits and limitations of the solutions that were adopted. Improvements to the methods are suggested and extensions of scope that seem promising in the light of the results of the thesis are examined.

9.1 Scope of the thesis

Event recognition is a central issue in many real-world problems. This thesis proposes a simple graphical representation, which models explicitly the temporal relations between events in an event recognition model. The main assumptions made in the representation are the following:

- The event recognition model is symbolic and events are defined hierarchically as temporal sequences of subevents. Such a hierarchy can be represented by a directed acyclic graph: the temporal classification network (TCN).
- The events at the lowest level of the hierarchy are assumed to be recognised by a separate system. This low-level event recognition system provides the interface between the digital signals and the TCN.
- The output of the low-level event recognition system is a stream of time-stamped occurrences of events at the lowest level of the TCN hierarchy. Event occurrences

are allowed to overlap.

- There are only three types of event defined in a TCN: conjunctive, disjunctive and repeating.
- The definition of conjunctive events should specify the exact sequence of supporting subevents, none of which may be missing for the event to be recognised.
- Limited use of negation is made by including rejecting events in a conjunctive event definition.
- A repeating event is defined as a repetition of a single supporter. The number of repetitions is bounded above and below.
- Each event definition contains a number of numeric temporal constraints. These constraints specify upper and lower bounds on the duration of the subevents and the distance between them, when the subevents are temporally related.

The main theme of the thesis is the development of refinement methods for setting the temporal parameters in a TCN model on the basis of limited data. Three refinement methods were presented in chapters 4, 5 and 6 respectively, each with a different scope. The first method makes use of a space-efficient memory scheme, but is dependent on the order of presentation of the examples. The second extends the memory scheme to avoid this problem and also uses heuristics to deal with noise in the data. Both these methods assume full supervision, i.e., training feedback for all of the events in the TCN. The third method relaxes this assumption to partial supervision, whereby training feedback is provided for a subset of the events. The following are further important assumptions made by the three refinement methods:

- No rejecting subevents are used in event definitions.
- The bounds on the number of repetitions in repeating events are fixed.
- The sequences of repeated subevent occurrences, which are used in the recognition of repeating events must be unbroken, i.e., no occurrence can be skipped.
- The expert needs to provide information about the relative weight between positive and negative examples and between the model-based and data-based fitness functions. The expert should also specify the maximum displacement, i.e., parameter window, for each parameter range.
- The model-based fitness function, i.e., proximity, treats the marginal change in all parameter ranges in a uniform way. For instance a unit's change in any of the duration

constraints is equivalent to a unit's change in any of the distance constraints.

- The feedback allocation algorithm used under partial supervision makes local decisions for each event in the TCN. Thus, it ignores the dependence between events which share the same supporting subevent.

The focus of this work has been on the development of the refinement algorithms, rather than the optimal choice of heuristic functions. The chosen functions are simple and intuitive, but their theoretical properties have not been examined in detail. A thorough theoretical analysis could reveal further implicit assumptions in the choice of the functions.

The methods proposed in the thesis have been applied to the thematic analysis of humpback whale songs. A TCN representation of the song structure was used and a small number of songs were encoded and used as training and test data. The aim here is to test the applicability of the representation to a real problem and evaluate the performance of the refinement methods. Some of the assumptions underlying the experimental setup are:

- The model of the song is for one season and one population of animals.
- Exceptional components of the song need to be identified and modelled explicitly.
- Two types of unit classification were used. The first, context-sensitive, assumes that units belonging to different themes can be distinguished even when their signal properties are very similar. The second, context-free, takes a more realistic approach, grouping context-sensitive units into context-free unit types. The important assumption is that this mapping is many-to-one, i.e., all occurrences of a context-sensitive unit belong to the same context-free unit type.
- The goal of the refinement in all of the experiments was to generalise an overly specific initial model.

9.2 Challenging issues and solutions

9.2.1 Event recognition

The event recognition scenario examined in the thesis differs in many respects from the standard approach to event recognition. The main difference is in the symbolic nature of the input data. The data is presented to the system as a sequence of time-stamped events, rather than the common approach of measuring a set of signal properties at fixed time intervals. This format

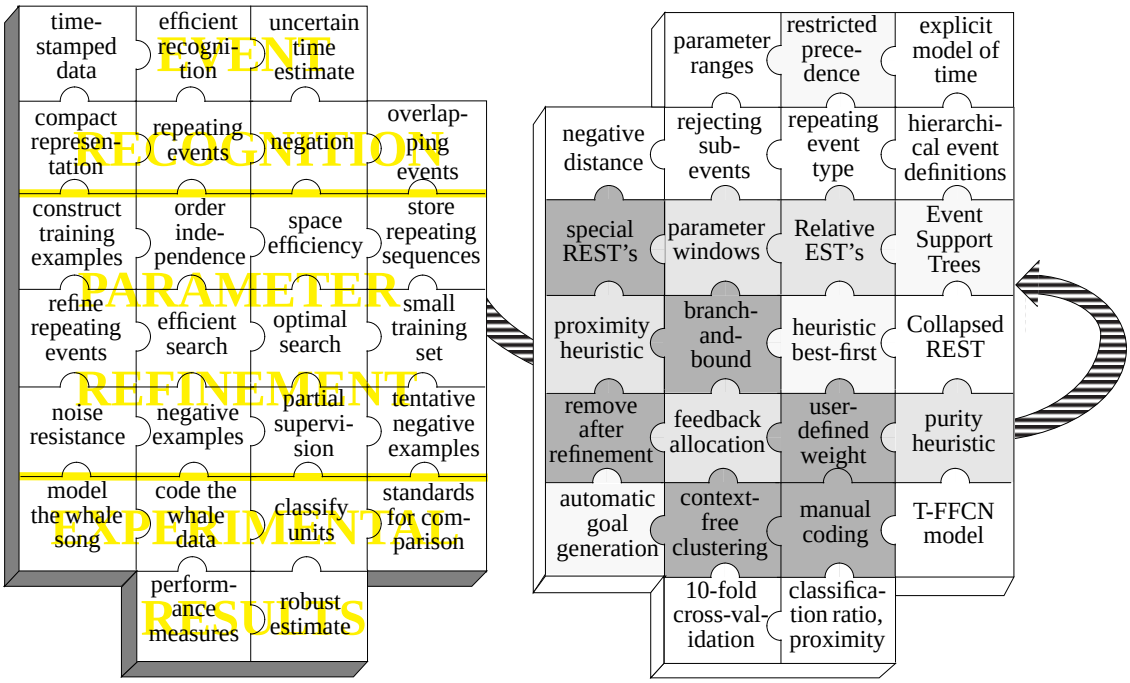


Figure 9.1: Graphical summary of challenging issues and solutions. The work presented in the thesis is separated into three parts: event recognition, parameter refinement and experimental results. Each jigsaw piece corresponds to an interesting technical issue in one of the three areas. The reverse of each piece shows the solution that was adopted and the extent to which it is judged adequate. The lighter the shade of the piece the more satisfactory the proposed solution.

removes the need for numerical modelling of the physical properties of events, other than their temporal properties. On the other hand, time is decoupled from the event recognition system and requires explicit modelling. Time is modelled in a TCN with the use of constraints on the duration of events and the distance between temporally related events. The type of temporal relation and the constraints that are used are very restricted in comparison to more general logics, used in other temporal event recognition systems. This restriction is considered necessary for preserving recognition efficiency. Some of the restrictive assumptions in the temporal aspects of the TCN may not be appropriate for all event recognition problems and the investigation of extended representations is of interest. An issue which is considered particularly important in the proposed temporal model is the uncertainty of time measurements in the input data. This may be either due to the nature of the events that are modelled or due to measurement errors. For this reason all temporal constraints are numerical ranges, rather than single values.

An additional consideration in the design of the TCN representation is the compactness of the model. The use of hierarchical event definitions allows the construction of compact event recognition models, where an event can be used as a subevent in several higher-order event definitions. This hierarchical support order of events in a TCN can be presented as a graphical model, which gives further insight in the structure of the model. A special type of event, which can cause a large increase in the size of a basic TCN model, i.e., a model using only conjunction and disjunction, is the repeating event. Repeating events cannot be dealt with naturally using simple conjunctive and disjunctive event definitions, as they involve long sequences of subevent occurrences which may also vary in length. In order to maintain the compactness of the representation, a special type of definition is used for repeating events, where the supporting subevent is allowed to precede itself. In other words, a repeating event is modelled as a sequence of occurrences of a single subevent, with similar temporal characteristics. A further extension to the basic TCN representation is the use of negation in conjunctive event definitions. Negation appears in the form of rejecting subevents, which are always temporally associated with a supporting subevent. This restriction prevents the definition of events which would fire continuously when their subevent did not fire. Finally, another important aspect of the TCN representation, which distinguishes it from the common approaches to event recognition, is the handling of overlapping events. Overlapping events can be naturally modelled with the use of negative distance constraints in the model. This is a further advantage of the explicit handling of time.

9.2.2 Parameter refinement

Due to the peculiarities of this event recognition scenario, the task of model refinement poses a number of challenging questions, which are not answered by standard refinement methods. The first such issue is the construction of training examples from the given data. The training data consists of a stream of low-level, input, event occurrences and a parallel stream of high-level, feedback, occurrences. The correspondence of sequences of input occurrences to the high-level occurrences is not provided and is not one-to-one, i.e., more than one sequence of input event occurrences may cause the high-level event to fire. The proposed solution to this problem is the construction of Event Support Trees (ESTs), which enumerate the possible supporting sequences for each high-level event occurrence. A similar process takes place

for negative examples, i.e., sequences which could have incorrectly led to the recognition of high-level events. An important limitation of this enumerative approach is that the number of alternative supporting sequences can increase exponentially, especially for negative examples. The solution initially adopted to this problem was to incrementally construct only the parts of the EST that were needed in the refinement. This approach worked well for the incremental refinement method (ORA),[1] described in chapter 4.

However, this method was shown to be dependent on the order of presentation of the examples and an extended memory scheme, based on the idea of the EST has been devised. This scheme accumulates information from a set of ESTs into a common structure called a Relative Event Support Tree (REST), which allows batch processing of the training data for each event. The problem of exponential size increase reappeared with the use of the REST and has been only partially solved by the use of maximum displacement windows for each parameter range in the model. These windows allow the pruning of the REST to a more manageable size. The specification of the parameter windows is not straightforward and they do not solve the problem completely. Therefore, the development of alternative memory schemes for order-independent refinement is still an open issue. One approach would be to prune less promising supporting sequences from the constructed ESTs, before adding them to the REST. The cost functions used in the incremental refinement algorithm could be of use in this process. This idea has not been investigated in the thesis. An additional problem with the REST memory scheme is the handling of repeating events. The approach taken in the thesis is to treat repeating sequences as conjunctions of their component event occurrences. A small modification of the REST is needed to accommodate this approach. However, this is an unnatural way to handle repeating events and results in very large RESTs. During refinement, these RESTs are collapsed into rectangular areas in a two-dimensional parameter space. Thus, it seems possible that a similar approach could be taken in the initial storage of the examples for repeating events. This would decrease the storage requirements of the memory scheme.

Refinement of the model parameters involves a search for a good set of parameters using the data and the original model. The choice of search method is affected mainly by two criteria: optimality and efficiency. Ideally, an efficient search method is sought, which also guarantees the optimal solution, according to the search criteria. Such a method has been developed for the

[1] ORA: Optimal Refinement Algorithm.

restricted memory scheme. It performs a branch-and-bound search, which guarantees the optimal solution, and in practice avoids exhaustive enumeration. In the extended memory scheme, a heuristic alternative (LRA)[2] was preferred, because the search space is more complex. This alternative adopts a heuristic best-first search, based on a top-down traversal of the REST. The dependence of the search method on the structure of the REST causes brittleness to some extent and an alternative method, which would have a more global view of the REST, might be preferable. However, it seems that such an approach would be associated with higher computational costs. This is another open issue.

The fitness function used in the evaluation of candidate solutions combines two heuristic measures: proximity and purity. Proximity measures the displacement of the candidate solution from the original parameter values. It is used as a model bias of minimum change which is necessary due to the small size of the training set. Purity is a measure of the positive and negative coverage of the candidate solution. Thus, it assesses the classification accuracy of the solution in the training set. Overtraining to noise in the data is avoided by the use of the proximity bias and the tolerance of less than perfect classification. The chosen proximity, purity and combined fitness functions are very simple and their properties have not been examined in detail. This is another area were further work is necessary. Especially problematic is the treatment of negative examples, which can affect significantly the performance of the refinement algorithm. All sequences of low-order events, which can cause the incorrect recognition of a high-order one, are used as negative examples. The number of such sequences can be very large in some cases, threatening to dominate the search. The adopted solution to this problem is the rescaling of positive and negative examples and the automatic removal of negative examples which should not affect the search. An alternative method, which would assess the importance of negative examples in the search for good solutions, would be preferable.

Refinement under partial supervision introduces additional difficulties. Training feedback is provided only for the output nodes of the TCN and a method is needed to deduce the classification of intermediate, hidden, events. The adopted solution is a feedback allocation algorithm (RAPS),[3] which uses model-based, i.e., proximity, information to guide the allocation of feedback to the hidden events. The feedback allocation algorithm combines a number of heuristic functions, providing evidence accumulation and feedback distribution at different stages. The

[2]LRA: Lazy Refinement Algorithm
[3]RAPS: Refinement Algorithm under Partial Supervision.

choice of these functions is again based on intuition rather than sound theoretical analysis and there is room for improvement. Among the technical problems faced in the design of the feedback allocation algorithm, one which is of particular importance is the treatment of negative examples. Due to the fact that there is no direct feedback for each event in the model, there is a large number of events which are tentatively recognised until training feedback is received and allocated through the network. The first problem caused by this approach is an increase in the size of the RESTs, which is not dealt with in the thesis. A further problem is the fact that most of these events are negative examples, which can be excluded in a large number of ways. Since the feedback allocation algorithm makes local decisions at each node of the TCN, the danger of placing a higher weight than necessary on negative examples increases. A partial solution to this problem is adopted, which ignores event occurrences supporting higher-order events when the latter have been excluded by refinement. Further work is required in this area, aiming at more informed decisions in the allocation of feedback.

9.2.3 Experimental results

The proposed methods have been applied successfully to a real-world problem. The structure of the humpback whale song is hierarchical, consisting of units, subphrases, phrases and themes. Thus, the TCN is a suitable modelling method for the song. The temporal structure of the song is also easily represented in the TCN and the resulting model has been shown to provide robust event recognition, when the temporal parameters are set correctly. One important aspect of the TCN, which is not tested with the whale song data, is the effect of overlapping events. Some suggestions have been made in chapter 7 to introduce this feature artificially to the problem, but they have not been implemented.

The transcription of the whale song data used in the experiments suffers in two ways. First, the low-level event occurrences, units, have been coded manually, using photocopies of spectrogram logs. This method is bound to introduce some inaccuracy and does not allow the transcription of a large quantity of data. A large data set would be desirable to test the performance of the system. The second problem in the transcription process is the classification of units. The initial transcription has been done using context-sensitive units. This is clearly an unrealistic assumption and an attempt to relax it, by grouping the context-sensitive units into context-free units types, was made. These unit types are defined in terms of signal properties

and could be used as a naive unit classifier. If this was the way they were used, however, they would not provide the discrimination between units, which is assumed by the many-to-one mapping between context-sensitive units and context-free unit types. Proper treatment of the data requires the digitisation of the recordings and the automatic extraction of properties from the digital signal. In addition, a context-free unit classifier should be used to extract the training data.

Four experiments have been performed with the whale song data, evaluating the performance of the full and partial supervision refinement algorithms with context-sensitive and context-free data. Due to the shortage of data, the evaluation of the performance of the refinement methods presents a further challenge. The adopted solution was to perform a 10-fold cross-validation, varying also the degree of difficulty for the refinement task. A simple problem-specific algorithm has been developed which generates the initial model from data and can also be used to generate an approximation of the goal model after refinement. The initial model is overly specific and the refinement involves only generalisation. The degree of difficulty for the refinement varies with the amount of data used to build the initial model. The generalisation-only refinement task is a shortcoming of the experimental approach. A different approach, which overcomes this problem, would be to start with the perfect model and add a variable amount of random noise. An additional advantage of this alternative is that the target for the refinement is known exactly, while in the adopted approach it can only be approximated by the model-construction algorithm. The refined model is evaluated using both data-based, i.e., classification on unseen data, and model-based, i.e., proximity, performance measures. According to both criteria the performance of the methods is encouraging.

9.3 Extending the scope

As explained above, there is room for improvement in many aspects of the proposed methods. Some suggestions in this direction have been given in the previous section. Additionally, the modelling and refinement approaches could be extended to become applicable to a wider class of problems. This section examines interesting extensions of this type.

One of the restricting assumptions in the TCN representation is the rigidness of logical definitions. For example, the requirement to recognise all of the supporting subevents of a conjunctive event may lead to problems when a low-level event is not recognised due to noise in

the signal. This would cause a chain of recognition failures in the hierarchy of event definitions. An alternative approach would be to use probabilistic evidence combination, which is a popular method of dealing with classification under uncertainty. In the case of the TCN, conjunctive events would perform a weighted combination of evidence on the recognition of their supporters and the output would be a belief measure on the recognition of the event. The weights used in the combination function would reveal the dependence of the higher-order event on its supporter. This idea would also facilitate the development of a probabilistic feedback allocation method, since the TCN could be treated as a probabilistic graphical model. One drawback of this approach is the specification of probabilities in the model. If these are to be learned from data, the assumption of a small training set needs to be relaxed.

Removing the restriction of a small data set would also allow the extension of the refinement method to deal with model restructuring. The goal of restructuring is to improve the classification performance of the system, by modifying the structure of the TCN, i.e., removing and adding nodes and links in the network. One way to approach this problem would be to use the results of the parameter refinement search as a guide for promising structural modifications. For example, if the parameter change required to cover a positive example is too costly or even impossible due to negative examples, a structural change of the event definition might yield better results. This approach can be naturally combined with the probabilistic classification model, mentioned above. For example, thresholds for the dependence weights could be introduced, below which the corresponding link would be removed.

Another interesting extension is to allow recognition information to be fed back to the low-level event recognition system. This feedback can take place both in the recognition and in the refinement stages. During recognition it would allow contextual information to be taken into account in the recognition of low-level events. Whether or not this approach is useful will depend on the importance of correct classification at this low level. In the refinement stage the feedback could suggest changes to the low-level event recognition system, which would improve the overall recognition performance. An example would be the search for an event which has not been recognised due to noise in the signal, but is needed for the recognition of a higher-order event. This interaction between the symbolic and signal processing levels of event recognition systems has also been noted by the developers of the blackboard-based system IPUS [51].

Some event recognition tasks require the integration of a temporal with a non-temporal model. An example of this type of task is the integration of grammar and vocabulary in a speech recognition system. As mentioned in chapter 3, the TCN can be seen as a simple grammar, which is augmented with temporal constraints. Therefore, the representation could be extended to facilitate an integrated model, by allowing non-temporal event definitions. Thus, the concept of a noun could be defined as the disjunction of all nouns in the vocabulary. Using such disjunctive definitions as an interface, a TCN can be used to model the words of the vocabulary and a non-temporal classification network can provide the grammar model.

Finally, it is clear that further evaluation of the proposed methods is needed. The good performance of the methods on one example application does not prove their applicability in general. The evaluation on a problem where overlapping events are used would be particularly interesting.

9.4 Summary

This thesis has explored the task of refining the temporal parameters of a hierarchical event recognition model using a small data set. The solution involves two important features: a new graphical representation for the event recognition model and novel parameter refinement methods that are tailored to this representation. These ideas have been implemented in a system which was evaluated on a real-world task: the thematic analysis of songs of the humpback whale. The results of this experiment were encouraging, showing that the system is able to improve significantly an initially inaccurate model, even with the use of very limited training data. Restrictions of the methods have been indicated, which present opportunities for improvement. In addition, the proposed methods can be extended in various ways to become applicable to a wider range of related problems. Overall, the task of event recognition has been viewed in a novel way and plausible solutions to the challenging issues that arise have been provided. Moreover, the positive results on the whale song problem suggest the practical utility of the methods.

Numerous challenging questions have been generated in the course of the work presented in this thesis and a large number of them remain unanswered. This observation suggests that event recognition and in particular the automatic construction of event recognition systems from data is an exciting new research area.

Bibliography

[1] *Proceedings of the National Conference on Artificial Intelligence*. Morgan-Kaufmann, August 1986.

[2] J. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23:123–154, 1984.

[3] J. F. Baldwin, R. M. Gooch, and T. P. Martin. Fuzzy Processing of Hydrophone Sounds. *Fuzzy Sets and Systems*, 77:35–47, 1996.

[4] R. Barlow. Event Classification Using Weighting Methods. *Computer Physics Communications*, 72:202–219, 1987.

[5] L. V. Belyaev and L. P. Falcone. Noise-Resistant Classification: subsymbolic and hybrid architectures for event classification in plasma physics. In IWML91 [44], pages 581–585.

[6] Y. Bengio and P. Frasconi. Credit Assignment through Time: alternatives to backpropagation. In *Advances in Neural Information Processing Systems*, volume 6, pages 157–166, 1994.

[7] F. Bergadano and A. Giordana. Guiding Induction with Domain Theories. In Y. Kodratoff and R. Michalski, editors, *Machine learning: an artificial intelligence approach*, volume III, pages 474–492. Morgan-Kaufmann, San Mateo, CA, 1990.

[8] M. Blaquiére and F. Evrard. Recognition of Temporal Phenomena in Petroleum Processes. In *Proceedings of the Conference on Knowledge Based Expert Systems for Engineering: Classification Education and Control*, pages 255–269. Computational Mechanics Publications, August 1987.

[9] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[10] B. G. Buchanan and E. H. Shortliffe, editors. *Rule-Based Expert Systems: the MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, 1984.

[11] D. Chabot. A Quantitative Technique to Compare and Classify Humpback Whale (Megaptera Novaeangliae) Sounds. *Ethology*, 77:89–102, 1988.

[12] P. J. Clapham and D. K. Mattila. Humpback Whale Songs as Indicators of Migration Routes. *Marine Mammal Science*, 6(2):155–160, 1990.

[13] C. Clark. The Acoustic Repertoire of the Southern Right Whale, a Quantitative Analysis. *Animal Behavior*, 30:1060–1071, 1982.

[14] C. Clark, P. Marler, and K. Beeman. Quantitative Analysis of Animal Vocal Phonology: an application to swamp sparrow song. *Ethology*, 76:101–115, 1987.

[15] C. W. Clark, W. T. Ellison, and K. Beeman. Acoustic Tracking of Migrating Bowhead Whales. In *Proceedings of OCEANS 86*, pages 341–346. IEEE Ocean Engineering Society, 1986.

[16] P. Clark and T. Niblett. The CN2 Algorithm. *Machine Learning*, 3(4):261–283, 1989.

[17] W. W. Cohen. Abductive Explanation-Based Learning: a solution to the multiple inconsistent explanation problem. *Machine Learning*, 8:167–219, 1992.

[18] S. Craw and D. Sleeman. The Flexibility of Speculative Refinement. In IWML91 [44], pages 28–32.

[19] A. Czyżewski and A. Kaczmarek. Speech Recognition Systems based on Rough Sets and Neural Networks. Technical Report ICS-60/94, Institute of Computer Science, Warsaw University of Technology, Warsaw, Poland, 1994.

[20] R. De Mori, L. Lam, and M. Gilloux. Learning and Plan Refinement in a Knowledge-Based System for Automatic Speech Recognition. In Waibel and Lee [107], pages 246–261.

[21] R. De Mori and Y. F. Mong. A System of Plans for Connected Speech Recognition. In *Proceedings of the National Conference on Artificial Intelligence*, pages 92–95. Morgan-Kaufmann, 1984.

[22] T. G. Dietterich. Learning at the Knowledge Level. *Machine Learning*, 1(3):287–316, 1986.

[23] S. K. Donoho and L. A. Rendell. Rerepresenting and Restructuring Domain Theories: a constructive induction approach. *Journal of Artificial Intelligence Review*, 2:411–446, 1995.

[24] E. Dorken, E. Milios, and S. H. Nawab. Knowledge-Based Signal Processing Applications. In Oppenheim and Nawab [73], pages 303–331.

[25] C. Dousson. Alarm Driven Supervision for Telecommunication Network II - on-line chronicle recognition. *Annales des Télécommunication*, 51(9-10):501–508, 1996.

[26] C. Dousson, P. Gaborit, and M. Ghallab. Situation Recognition: representation and algorithms. In IJCAI93 [43], pages 166–172.

[27] T. Ellman. Explanation-Based Learning: a survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163–221, 1989.

[28] R. Engelmore and T. Morgan, editors. *Blackboard Systems*. Addison-Wesley, 1988.

[29] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy. The Hearsay-II Speech-Understanding System: integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):214–253, 1980.

[30] A. S. Frankel. *Acoustic and Visual Tracking Reveals Distribution, Song Variability and Social Roles of Humpback Whales in Hawaiian Waters*. PhD thesis, Oceanography Department, University of Hawaii, 1994.

[31] K. T. Frantzi, T. Panayiotopoulos, and C. D. Spyropoulos. Extending Allen's Relations for Uncertain Time Points and Uncertain Intervals. In *Proceedings of the 7th Irish Conference on Artificial Intelligence and Cognitive Science*, 1994.

[32] P. Frasconi, M. Gori, and G. Soda. Recurrent Neural Networks and Prior Knowledge for Sequence Processing: a constrained nondeterministic approach. *Knowledge Based Systems*, 8(6):313–332, 1995.

[33] P. Frumhoff. Aberrant Songs of Humpback Whales (Megaptera Novaeangliae): clues to the structure of humpback songs. In Payne [77], pages 81–127.

[34] A. Ginsberg, S. M. Weiss, and P. Politakis. Automatic Knowledge Base Refinement for Classification Systems. *Artificial Intelligence*, 35(2):197–226, 1988.

[35] R. P. Gorman and T. J. Sejnowski. Learned Classification of Sonar Targets using a Massively Parallel Network. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(7):1135–1140, 1988.

[36] N. Graner and D. Sleeman. MUSKRAT: a multistrategy knowledge refinement and acquisition toolbox. In *Proceedings of the International Workshop on Multi-Strategy Learning*, pages 107–119. Center for Artificial Intelligence, George Mason University, May 1993.

[37] L. N. Guinee, K. Chu, and E. M. Dorsey. Changes Over Time in the Songs of Known Individual Humpback Whales (Megaptera Novaeangliae). In Payne [77], pages 59–80.

[38] L. N. Guinee and K. B. Payne. Rhyme-like Repetitions in Songs of Humpback Whales. *Ethology*, 79:295–306, 1988.

[39] G. W. Hafner, C. L. Hamilton, W. W. Steiner, T. J. Thompson, and H. E. Winn. Signature Information in the Song of the Humpback Whale. *Journal of the Acoustical Society of America*, 66(1):1–6, 1979.

[40] J. D. Hamilton. A new Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle. *Econometrica*, 57(2):357–384, 1989.

[41] P. D. Hewitt, P. J. C. Skitt, and R. C. Witcomb. A Self-Organising Feedforward Network Applied to Acoustic Data. In J. G. Taylor and C. L. T. Mannion, editors, *New Developments in Neural Computing*, pages 71–78. Adam Hilger, 1989.

[42] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[43] *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1993.

[44] *Proceedings of the International Workshop on Machine Learning*. Morgan-Kaufmann, June 1991.

[45] R. M. Keller. Defining Operationality for Explanation-Based Learning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 482–487. Morgan-Kaufmann, 1987.

[46] S. Kockskämper, B. Neuman, and M. Schick. Extending Process Monitoring by Event Recognition. In *Proceedings of the 2nd International Conference on Intelligent Systems Engineering*, pages 455–460, 1994.

[47] B. Kostek. Automatic Reasoning about Acoustic Data: problems with preprocessing, classification and decision uncertainty. In *Proceedings of the International Symposium on Intelligent Data Analysis*, pages 99–103, 1995.

[48] K. Kumar and A. Mukerjee. Temporal Event Conceptualization. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1987.

[49] R. Lacher. Node Error Assignment in Expert Networks. In A. Kandel and G. Langholz, editors, *Hybrid Architectures for Intelligent Systems*, pages 29–48. CRC Press, BOCA RATON, 1992.

[50] P. Langley. *Elements in Machine Learning*. Morgan-Kaufmann, San Mateo, CA, 1995.

[51] V. R. Lesser, S. H. Nawab, and F. I. Klassner. IPUS: an architecture for the integrated processing and understanding of signals. *Artificial Intelligence*, 77:129–171, 1995.

[52] R. P. Lippmann. Review of Neural Networks for Speech Recognition. In Waibel and Lee [107], pages 374–391.

[53] B. Lowerre and R. Reddy. The HARPY Speech Understanding System. In Waibel and Lee [107], pages 576–586.

[54] J. J. Mahoney and R. J. Mooney. Combining Connectionist and Symbolic Learning to Refine Certainty-Factor Rule Bases. *Connection Science*, 5:339–364, 1993.

[55] D. K. Mattila, L. N. Guinee, and C. A. Mayo. Humpback Whale Songs on a North Atlantic Feeding Ground. *Journal of Mammalogy*, 68(4):880–883, 1987.

[56] D. McDermott. A Temporal Logic for Reasoning about Processes and Plans. *Cognitive Science*, 6:101–155, 1982.

[57] D. J. McSweeney, K. C. Chu, W. F. Dolphin, and L. N. Guinee. North Pacific Humpback Whale Songs: a comparison of southeast Alaskan feeding ground songs with Hawaiian wintering ground songs. *Marine Mammal Science*, 5(2):139–148, 1989.

[58] D. K. Mellinger and C. W. Clark. A Method for Filtering Bioacoustic Transients by Spectrogram Image Convolution. In *Proceedings of OCEANS 93*, volume 3, pages 122–123. IEEE Ocean Engineering Society, 1993.

[59] R. S. Michalski. A Theory and Methodology of Inductive Learning. In Michalski et al. [61], pages 83–138.

[60] R. S. Michalski, I. Mozetič, J. Hong, and N. Lavrac. The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In AAAI86 [1], pages 1041–1045.

[61] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors. *Machine learning: an artificial intelligence approach*, volume I. Morgan-Kaufmann, Palo Alto, CA, 1983.

[62] E. Milios and S. H. Nawab. Signal Abstraction Concept for Signal Interpretation. In Oppenheim and Nawab [73], pages 303–331.

[63] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[64] T. M. Mitchell. Version Spaces: a candidate-elimination approach to rule learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 305–310, 1977.

[65] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-Based Generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[66] D. J. Montana. Empirical Learning using Rule Threshold Optimization for Detection of Events in Synthetic Images. *Machine Learning*, 5:427–450, 1990.

[67] R. J. Mooney. Explanation Generalisation in EGGS. In B.G. Buchanan and D.C. Wilkins, editors, *Readings in knowledge acquisition and learning*, pages 556–575. Morgan-Kaufmann, San Mateo, CA, 1993.

[68] R. A. Morris, W. D. Shoaff, and L. Khatib. Path Consistency in a Network of Non-convex Intervals. In IJCAI93 [43], pages 655–660.

[69] T. Niblett and I. Bratko. Learning Decision Rules in Noisy Domains. In *Research and Development in Expert Systems*, volume 3, pages 25–34. Cambridge University Press, 1987.

[70] H. P. Nii, E. A. Feigenbaum, J. J. Anton, and A. J. Rockmore. Signal-to-Symbol Transformation: HASP/SIAP case study. In Engelmore and Morgan [28], pages 135–157.

[71] C. E. Nishimura and D. M. Conlon. IUSS Dual Use: monitoring whales and earthquakes using SOSUS. *Marine Technology Society Journal*, 27(4):13–21, 1993.

[72] C. W. Omlin, K. K. Thornber, and C. L. Giles. Fuzzy Finite-State Automata can be Deterministically Encoded into Recurrent Neural Networks. Technical Report CS-TR-3599 and UMIACS-96-12, Department of Computer Science, University of Maryland, Maryland, USA, 1996.

[73] A. V. Oppenheim and S. H. Nawab, editors. *Symbolic and Knowledge-Based Signal Processing*. Prentice Hall, New Jersey, 1992.

[74] Z. Pawlak, J. Grzymala-Busse, R. Slowinski, and W. Ziarko. Rough Sets. *Communications of the ACM*, 38(11):89–95, 1995.

[75] K. Payne and R. Payne. Large Scale Changes over 19 Years in Songs of Humpback Whales in Bermuda. *Zeitschrift für Tierpsychologie (Ethology)*, 68:89–114, 1985.

[76] K. Payne, P. Tyack, and R. Payne. Progressive Changes in the Songs of Humpback Whales (Megaptera Novaeangliae): a detailed analysis of two seasons in Hawaii. In Payne [77], pages 9–57.

[77] R. Payne, editor. *Communication and Behaviour of Whales*. Westview Press, Boulder, CO, 1983.

[78] R. Payne and L. N. Guinee. Humpback Whale (Megaptera Novaeangliae) Songs as an Indicator of "Stocks". In Payne [77], pages 333–358.

[79] R. S. Payne and S. McVay. Songs of Humpback Whales. *Science*, 173(3997):585–597, 1971.

[80] M. J. Pazzani and C. A. Brunk. Detecting and Correcting Errors in Rule-Based Expert Systems: an integration of empirical and explanation-based learning. *Knowledge Acquisition*, 3:157–173, 1991.

[81] J. R. Potter, D. K. Mellinger, and C. W. Clark. Marine Mammal Call Discrimination using Artificial Neural Networks. *Journal of the Acoustical Society of America*, 96(3):1255–1262, 1994.

[82] J. R. Quinlan. Learning Efficient Classification Procedures and Their Application to Chess End Games. In Michalski et al. [61], pages 463–482.

[83] J. R. Quinlan. The Effect of Noise in Concept Learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine learning: an artificial intelligence approach*, volume II, pages 149–166. Morgan-Kaufmann, Palo Alto, CA, 1986.

[84] J. R. Quinlan. Learning Logical Definitions from Relations. *Machine Learning*, 5:239–266, 1990.

[85] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, San Mateo, CA, 1993.

[86] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *IEEE Proceedings*, 77(2):257–285, 1989.

[87] L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE Magazine on Acoustics, Speech and Signal Processing*, 3(1):4–16, 1986.

[88] P. S. Rosenboom and J. E. Laird. Mapping Explanation-Based Generalisation onto Soar. In AAAI86 [1], pages 561–567.

[89] D. Sarantis. *Model-Based Detection of Man-Made Structures in Sequences of Airport Aerial Images*. PhD thesis, Department of Electrical Engineering, University of Manchester, Manchester, UK, 1996.

[90] W. E. Sarrett and M. J. Pazzani. One-Sided Algorithms for Integrating Empirical and Explanation-Based Learning. In *Proceedings of the International Workshop on Machine Learning*, pages 26–28. Morgan-Kaufmann, June 1989.

[91] J. C. Schlimmer and D. Fisher. A Case Study of Incremental Concept Induction. In AAAI86 [1], pages 496–501.

[92] T. J. Sejnowski and C. R. Rosenberg. Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, 1:145–168, 1987.

[93] Y. Shoham. *Reasoning about Change*. MIT Press Series in Artificial Intelligence. MIT Press, Cambridge, MA, 1988.

[94] E. H. Shortliffe and B. G. Buchanan. A Model of Inexact Reasoning in Medicine. In Buchanan and Shortliffe [10], pages 233–262.

[95] P. J. C. Skitt, M. A. Javed, S. A. Sanders, and A. M. Higginson. Process Monitoring using Auto-Associative Feed-Forward Artificial Neural Networks. *Journal of Intelligent Manufacturing*, 4:79–94, 1993.

[96] D. Smith. *The Application of Artificial Neural Networks to the Classification of Underwater Cetacean Sounds*. PhD thesis, University of Exeter, 1993.

[97] G. G. Towell and J. W. Shavlik. Refining Symbolic Knowledge using Neural Networks. In *Proceedings of the International Workshop on Multi-Strategy Learning*, pages 257–272. Center for Artificial Intelligence, George Mason University, November 1991.

[98] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the National Conference on Artificial Intelligence*, pages 861–865. MIT Press, July 1990.

[99] P. Tyack. Interactions Between Singing Hawaiian Humpback Whales and Conspecifics Nearby. *Behavioral Ecology and Sociobiology*, 8:105–116, 1981.

[100] P. Tyack. Differential Response of Humpback Whales (Megaptera Novaeangliae), to Playback of Song or Social Sounds. *Behavioral Ecology and Sociobiology*, 13:49–55, 1983.

[101] P. E. Utgoff. Incremental Induction of Decision Trees. *Machine Learning*, 4(2):161–186, 1989.

[102] J. VanLeeuwen, editor. *Handbook of Theoretical Computer Science: Algorithms and Complexity*, volume A. Elsevier Science Publishers, 1990.

[103] L. Vila. A Survey on Temporal Reasoning in Artificial Intelligence. *Artificial Intelligence Communications*, 7(1):4–28, 1994.

[104] M. Vilain and H. Kautz. Constraint Propagation Algorithms for Temporal Reasoning. In AAAI86 [1], pages 377–382.

[105] A. Waibel, M. Finke, D. Gates, M. Gavalda, T. Kemp, A. Lavie, L. Levin, M. Maier, L. Mayfield, A. Mcnair, I. Rogina, K. Shima, T. Sloboda, M. Woszczyna, T. Zeppenfeld, and P. Zhan. JANUS-II - Translation of Spontaneous Conversational Speech. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, pages 409–412. IEEE Press, 1996.

[106] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme Recognition using Time-Delay Neural Networks. In Waibel and Lee [107], pages 393–404.

[107] A. Waibel and K. F. Lee, editors. *Readings in Speech Recognition*. Morgan-Kaufmann, 1990.

[108] A. Walker, R. Fisher, and N. Mitsakakis. Singing Maps: classification of whalesong units using a self-organising mapping algorithm. Technical Report DAI-833, Department of Artificial Intelligence, Edinburgh University, Edinburgh, UK, 1996.

[109] R. L. Watrous. Phoneme Discrimination using Connectionist Networks. *Journal of the Acoustical Society of America*, 87(4):1753–1772, 1990.

[110] M. A. Williams. Hierarchical Multi-Expert Signal Understanding. In Engelmore and Morgan [28], pages 387–415.

[111] H. E. Winn and N. E. Reichley. Humpback Whale, Megaptera Novaeangliae. In S. H. Ridgway and Sir R. Harrison, editors, *Handbook of Marine Mammals*, volume 3, pages 241–273. Academic Press, 1985.

[112] H. E. Winn, T. J. Thompson, W. C. Cummings, J. Hain, J. Hudnall, H. Hays, and W. W. Steiner. Song of the Humpback Whale: population comparisons. *Behavioral Ecology and Sociobiology*, 8:41–46, 1981.

[113] H. E. Winn and L. K. Winn. The Song of the Humpback Whale Megaptera Novaeangliae in the West Indies. *Marine Biology*, 47:97–114, 1978.

Appendix A

Inference algorithms

This appendix presents a detailed pseudocode description of the inference algorithm for temporal classification. The elastic temporal distance relation is used for defining precedence sequences in event definitions. Events are related to their subevents with the use of the following two predicates:

$\texttt{supports}(a_j, e_i, e_n, ((a_l, (s_j^-, s_j^+)), (d_j^-, d_j^+)))$,
$\texttt{rejects}(a_j, e_i, e_n, ((a_l, (s_j^-, s_j^+)), (d_j^-, d_j^+)))$,

where: e_n is the event being defined,
e_i the supporting/rejecting subevent,
a_j a unique label for this particular use of e_i in e_n, i.e., if e_i is used again in the definition of e_n it will have a different label,
a_l the label of the subevent which a_j `precedes` or '?' if a_j is terminal,
(s_j^-, s_j^+) the distance range between a_j and a_l or $(?, ?)$ if a_j is terminal,
(d_j^-, d_j^+) the duration range for e_i.

Thus, these two predicates include all the information for the corresponding subevent.

The main inference algorithm is presented in Alg. A.1 and corresponds to the narrative description given in Alg. 3.1. It receives a set of 0-order occurrences, updates their node histories and propagates the recognition forward through the TCN. This algorithm makes use of the **fires** and **fires-rep** algorithms which evaluate an event definition. These algorithms correspond to the **evaluate-definition** algorithm in Alg. 3.1. The former algorithm, presented in Alg. A.2, is responsible for conjunctive and disjunctive events, while the latter, presented in A.4, deals with repeating events. The **fires** algorithm allows for more than one potential terminal subevent in a conjunctive definition, i.e., temporally independent precedence sequences. Each such sequence is evaluated by the **fires-terminal** algorithm, Alg. A.3.

The **fires-terminal** algorithm presented in Alg. A.3 has a higher worst-case computational complexity than necessary. This is because it enumerates recursively all possible sequences of subevent occurrences. In order to improve its efficiency, the algorithm can be modified to work in a breadth-first manner, i.e., select all valid subevent occurrences before moving to the preceding subevent. Duplicate subevent occurrences can then be removed, as they correspond to the same higher-order event. In average, the branching factor of this search is expected to be quite low and the performance of the two alternatives similar. Recursive enumeration has been chosen here, simply because it is also used in the refinement algorithms in appendices D, E.

In addition to the inference algorithm, two pre-processing algorithms are presented. The **evaluate-distances** algorithm, Alg. A.5, checks the consistency of the temporal constraints,

ensuring partially that the qualitative constraint is not violated. This check is only partial, due to the use of constraint ranges, rather than single integer values. The conditions which need to be adhered to are the following:

1. A subevent cannot be preceded by two or more supporting ones.
2. A subevent cannot precede two or more others.
3. A rejecting subevent cannot be preceded.
4. Non-terminal subevents precede one other.
5. The maximum duration of the preceded subevent must be greater than the minimum distance from the preceding one, multiplied by -1.
6. A rejecting subevent cannot be terminal.
7. Disjunctive events should be supported only by terminal subevents.
8. Repeating events must be supported by a single subevent, which `precedes` itself. This is called *self-preceded subevent*.
9. A self-preceded subevent cannot be preceded by a different subevent.
10. A self-preceded subevent must support a repeating event.

The second pre-processing algorithm, **history-sizes**, Alg. A.6, calculates the history sizes of individual events, based on the relations in which their subevents participate. This is achieved by recognising the terminal subevent in a definition and propagating the temporal constraints back through the precedence sequence.

Input: F: list of new occurrences.

Output: F: updated list of all recognised events.

Globals: $G = (E, S, R)$: the model, in which E is the set of events in the partial support order, S the `supports` and R the `rejects` predicates; $\mathcal{H}$: the database of all local node histories.

TCN(F) **:**

$\forall e \in E$: % for each event, in partial order

% process each of the new supporting subevent occurrences: (usually there will be just one)

$\forall e_i(t_i^-, t_i^+) \in F, \exists$ `supports`$(a_j, e_i, e, ((a_l, s_j), (d_j^-, d_j^+))) \in S \wedge (t_i^+ - t_i^-) \in [d_j^- .. d_j^+]$:

`IF` **disjunctive**[a](e): $(f, T^-) \leftarrow (1, \{t_i^-\})$

`ELSEIF` **repeating**[b](e):

repetitions[c]$(e, (n^-, n^+))$, $(e_i, H_i) \in \mathcal{H}$

$H_i' \leftarrow H_i \backslash \{e_i(t_i^-, t_i^+)\}$ % remove terminal

$(f, T^-) \leftarrow$ **fires-rep**$(t_i^-, H_i', s_j, d_j, n^-, n^+, t_i^-, 1)$

$H_i' \leftarrow H_i \bigcup \{e_i(t_i^-, t_i^+)\}$ % add terminal again

`ELSEIF` $a_l = ?$: % terminal for conjunctive

$(f, T^-) \leftarrow$ **fires**(e, a_j, t_i^-)

`ELSE`: $(f, T^-) \leftarrow (-1, \{\})$

`ENDIF`

`IF` $f = 1$:

$\forall t^- \in T^-$:

$F \leftarrow F \bigcup \{e(t^-, t_i^+)\}$ % update list of firing nodes

`IF` $\neg$**disjunctive**(e):

$\mathcal{H} \leftarrow \mathcal{H} \backslash (e, H)$

$H \leftarrow H \bigcup \{e(t^-, t_i^+)\}$

$\mathcal{H} \leftarrow \mathcal{H} \bigcup (e, H)$ % update history

`ENDIF`

`ENDIF`

[a] Succeeds if the event is disjunctive.

[b] Succeeds if the event is repeating.

[c] Returns the repetition bounds.

Algorithm A.1: Pseudocode for the temporal classification algorithm.

Input: e: the conjunctive event; a_j: the terminal subevent; t_i^-: the start of the terminal subevent occurrence.

Output: f: flag of successful evaluation, i.e., recognition of e; T^-: set of alternative start points, if more than one sequences of subevents satisfy the constraints.

Globals: $G = (E, S, R)$: the model; $\mathcal{H}$: the database of all local node histories.

fires(e, a_j, t_i^-) **:**

$(f, T^-) \leftarrow$ **fires-terminal**(e, a_j, t_i^-, t_i^-)
`IF` $f = -1$: `RETURN` (f, T^-)
%Deal with temporally independent subevent sequences:
$\forall$ `supports`$(a_l, e_n, e, ((?, (?, ?)), (d_l^-, d_l^+))), a_l \neq a_j \wedge (e_n, H_n) \in \mathcal{H}$:
    $\forall e_n(t_n^-, t_n^+) \in H_n \wedge (t_n^+ - t_n^-) \in [d_l^- .. d_l^+]$:
        $H_n \leftarrow H_n \backslash \{e_n(t_n^-, t_n^+)\}$ % remove terminal
        $(f, T_n^-) \leftarrow$ **fires-terminal**(e, a_l, t_n^-, t_n^-)
        $H_n \leftarrow H_n \bigcup \{e_n(t_n^-, t_n^+)\}$ % add terminal again
        `IF` $f = -1$: `RETURN` (f, T_n^-) `ELSE`: $T^- \leftarrow T_n^- \bigcup T^-$

`RETURN` $(f = 1, T^-)$

Algorithm A.2: Pseudocode for the evaluation of conjunctive event definitions.

Input: e: the conjunctive event; a_j: the currently examined subevent in the sequence; t_i^-: the start of the subevent occurrence; t^-: the start of the occurrence of e, as calculated so far.

Output: f: flag of successful evaluation, i.e., recognition of e; T^-: set of alternative start points for e.

Globals: $G = (E, S, R)$: the model; $\mathcal{H}$: the database of all local node histories.

fires-terminal(e, a_j, t_i^-, t^-) **:**

$T^- \leftarrow \{\}$, $f \leftarrow -1$
`IF` ($\exists$ `supports`$(a_l, e_n, e, ((a_j, (s_n^-, s_n^+)), (d_n^-, d_n^+))) \in S \wedge (e_n, H_n) \in \mathcal{H}$):
    $\forall e_n(t_n^-, t_n^+) \in H_n, (t_n^+ - t_n^-) \in [d_l^- .. d_l^+] \wedge (t_i^- - t_n^+) \in [s_l^- .. s_l^+]$:
        `IF` $t_n^- < t^-$: $t^- \leftarrow t_n^-$
        $H_n \leftarrow H_n \backslash \{e_n(t_n^-, t_n^+)\}$ % remove occurrence
        $(f, T_n^-) \leftarrow$ **fires-terminal**(e, a_l, t_n^-, t^-)
        $H_n \leftarrow H_n \bigcup \{e_n(t_n^-, t_n^+)\}$ % add occurrence again
        `IF` $f = 1$: $T^- \leftarrow T_n^- \bigcup T^-$
    `IF` $f = -1$: `RETURN` $(f, T^- = \{\})$
`ELSE`: $f \leftarrow 1, T^- \leftarrow \{t^-\}$ % Start of precedence sequence
`ENDIF`
$\forall$ `rejects`$(a_l, e_n, e, ((a_j, (s_n^-, s_n^+)), (d_n^-, d_n^+))) \in R \wedge (e_n, H_n) \in \mathcal{H}$:
    `IF` ($\exists e_n(t_n^-, t_n^+) \in H_n \wedge (t_n^+ - t_n^-) \in [d_l^- .. d_l^+] \wedge (t_i^- - t_n^+) \in [s_l^- .. s_l^+]$):
        `RETURN` $(f = -1, T^- = \{\})$

`RETURN` (f, T^-)

Algorithm A.3: Pseudocode for the evaluation of a single precedence sequence.

Input: t_i^-: the start of the subevent occurrence; H_i: the local node history for the subevent; s_j, d_j: constraints on the subevent; n^-, n^+: the upper and lower limits of repetitions; t^-: the start of the event occurrence, as calculated so far; n: number of repetitions so far.

Output: f: flag of successful evaluation, i.e., recognition of e; T^-: alternative start points for e.

Globals: $G = (E, S, R)$: the model.

fires-rep$(t_i^-, H_i, s_j, d_j, n^-, n^+, t^-, n)$ **:**

$T^- \leftarrow \{\}, f \leftarrow -1$
IF $n > n^+$: RETURN $(f = -1, T^- = \{\})$
IF $(n < n^- \wedge \nexists e_i(t_n^-, t_n^+) \in H_n \wedge (t_n^+ - t_n^-) \in d_j \wedge (t_i^- - t_n^+) \in s_j)$:
    RETURN $(f = -1, T^- = \{\})$
$\forall e_i(t_n^-, t_n^+) \in H_n \wedge (t_n^+ - t_n^-) \in d_j \wedge (t_i^- - t_n^+) \in s_j$:
    IF $t_n^- < t^-$: $t^- \leftarrow t_n^-$
    $H_n \leftarrow H_n \backslash \{e_n(t_n^-, t_n^+)\}$ % remove occurrence
    $n_1 \leftarrow n + 1$
    $(f, T_n^-) \leftarrow$ **fires-rep**$(t_n^-, H_i, s_j, d_j, n^-, n^+, t^-, n_1)$
    $H_n \leftarrow H_n \bigcup \{e_n(t_n^-, t_n^+)\}$ % add occurrence again
    IF $f = 1$: $T^- \leftarrow T_n^- \bigcup T^-$
IF $f = -1$: RETURN $(f, T^- = \{\})$

RETURN (f, T^-)

Algorithm A.4: Pseudocode for the evaluation of a repeating event definition.

Input: $G = (E, S, R)$: the model.

Output: $\mathbb{T}/\mathbb{F}$.

evaluate-model$(G = (E, S, R))$ **:**

$\forall$supports$(a_j, e_n, e_i, (r_j, d_j)) \in S$:
    IF $\exists$supports$(a_l, e_m, e_i, ((a_j, s_l), d_l)) \in S$:
        IF ($\exists$supports$(a_l, e_m, e_i, ((a_j, s_l), d_l)) \in S \wedge a_l \neq a_k$): RETURN $\mathbb{F}$ %Rule 1
        IF $(d_j = (d_j^-, d_j^+) \wedge s_l = (s_l^-, s_l^+) \wedge d_j^+ < -s_l^-)$: RETURN $\mathbb{F}$ %Rule 5
    IF ($\exists$supports$(a_j, e_n, e_i, (r'_j, d'_j)) \in S \wedge r'_j \neq r_j \wedge d'_j \neq d_j) \vee$
        ($\exists$rejects$(a_j, e_n, e_i, (r'_j, d'_j)) \in R \wedge r'_j \neq r_j \wedge d'_j \neq d_j$): RETURN $\mathbb{F}$ %Rule 2
    IF $(r_j = (a_l, s_j) \wedge a_l \neq ?)$:
        IF **disjunctive**(e_i): RETURN $\mathbb{F}$ %Rule 7
        ELSEIF $\nexists$supports$(a_l, e_m, e_i, (r_l, d_l)) \in S)$: RETURN $\mathbb{F}$ %Rule 4
        ELSEIF $a_l = a_j$: %Rules 9,10
            IF ($\exists$supports$(a_k, e_m, e_i, (r_k, d_k)) \in S \wedge a_k \neq a_j$): RETURN $\mathbb{F}$
            ELSEIF $\neg$**repeating**(e_i): RETURN $\mathbb{F}$
        ELSEIF **repeating**(e_i): RETURN $\mathbb{F}$ %Rule 8

$\forall$rejects$(a_j, e_n, e_i, (r_j, d_j)) \in R$:
    IF ($\exists$supports$(a_l, e_m, e_i, ((a_j, s_l), d_l)) \in S \vee$
        IF $\exists$rejects$(a_l, e_m, e_i, ((a_j, s_l), d_l)) \in R$): RETURN $\mathbb{F}$ %Rule 3
    IF ($\exists$supports$(a_j, e_n, e_i, (r'_j, d'_j)) \in S \wedge r'_j \neq r_j \wedge d'_j \neq d_j$): RETURN $\mathbb{F}$ %Rule 2
    IF $(r_j = (a_l, s_j) \wedge \nexists$supports$(a_l, e_m, e_i, (r_l, d_l)) \in S)$: RETURN $\mathbb{F}$ %Rules 4+6
    IF **disjunctive**$(e_i) \vee$ **repeating**(e_i): RETURN $\mathbb{F}$ %Rules 7+8

RETURN $\mathbb{T}$

Algorithm A.5: Pseudocode for the evaluation of the model structure.

Input: $G = (E, S, R)$: the model.

Output: $HS = \{(e, t^+_{min}) | e \in E, t^+_{min} :$ distance to minimum end point of "useful" events $\}$.

history-sizes$(G = (E, S, R))$ **:**

$\forall e \in E$: $HS \leftarrow HS \bigcup \{(e, 0)\}$

$\forall$ `supports`$(a_j, e_n, e_i, ((?, ?), (d^-_j, d^+_j))) \in S \wedge \neg$**disjunctive**$(e_i)$):

  $HS \leftarrow$ **back-propagate-sizes**(a_j, e_i, d^+_j, HS, G)

$\forall$`supports`$(a_j, e_n, e_i, ((a_j, (s^-_j, s^+_j)), (d^-_j, d^+_j))) \in S \wedge$ **repetitions**$(e_i, (n^-, n^+))$:

  $HS \leftarrow HS \backslash \{(e_n, t^+_{min})\}$

  $HS \leftarrow HS \bigcup \{(e_n, (n^+ \times (s^+_j + d^+_j)))\}$

`RETURN` HS

back-propagate-sizes(a_j, e_i, d_j, HS, G)

`IF` $\exists$`supports`$(a_l, e_m, e_i, ((a_j, (s^-_l, s^+_l)), (d^-_l, d^+_l))) \in S \wedge (e_m, t^+_{min}) \in HS$:

  `IF` $s^+_l + d_j > t^+_{min}$:

    $HS \leftarrow HS \backslash \{(e_m, t^+_{min})\}$

    $HS \leftarrow HS \bigcup \{(e_m, s^+_l + d_j)\}$

  `ENDIF`

  $HS \leftarrow$ **back-propagate-sizes**$(a_l, e_i, (d^+_l + s_l + d_j), HS, G)$

`ENDIF`

$\forall$`rejects`$(a_l, e_m, e_i, ((a_j, (s^-_l, s^+_l)), d_l)) \in R \wedge (e_m, t^+_{min}) \in HS$:

  `IF` $s^+_l + d_j > t^+_{min}$:

    $HS \leftarrow HS \backslash \{(e_m, t^+_{min})\}$

    $HS \leftarrow HS \bigcup \{(e_m, (s^+_l + d_j))\}$

  `ENDIF`

`RETURN` HS

Algorithm A.6: Pseudocode for calculating maximum history sizes.

Appendix B

Search space for optimal specialisation

B.1 Distance-only specialisation

The task that was set for the specialisation of distance (section 4.4.2) involved the pruning of a negative Event Support Tree (EST), by pruning all the sequences that it contains. The goal of the algorithm is to efficiently arrive at the optimal, least-cost, set of model changes which prune the whole tree. An indicator of the complexity of this task is the number of possible choices that exist, i.e., the size of the search space for the algorithm. This in turn depends on the size of the EST and the interdependence of pruning choices, e.g. the fact that by pruning the most expensive of a set of sibling nodes, all other siblings are implicitly pruned, too. The size of the EST depends on two parameters:

a. The *height* of the tree h.

b. The *branching factor* b, which, for simplicity, is assumed to be uniform.

Given these parameters, the task is to calculate the size of the set of distinct solutions for pruning the tree. The additional simplifying assumption which is made here is that all specialisation takes place on the same side of the model. This set can be generated by the blind-search algorithm described in Alg. B.1. In brief, the algorithm generates the children of the node, n, proposes as a solution the pruning of the most expensive child, n_1, and then recursively calls itself for each of the children of n_1, to generate the alternative pruning solutions. Computationally, the most important point is that *all* of these alternative solutions, involving the children of n_1, use n_2, which is the next most expensive sibling of n_1, to prune the remaining siblings of n_1. When moving from n_2 to n_3 in the `FOR` loop, *all* but the first of the so far generated solutions contain n_2. As a result, **replace-node**(n_2, Z, z_j, n_3) applies to all of the solutions generated by **blind-search**(n_1) and generates a new alternative for each such solution. Thus, if $|Z_1|$ is the number of solutions generated by **blind-search**(n_1) and $|Z_2|$ that generated by **blind-search**(n_2), **replace-node**(n_2, Z, z_j, n_3) generates $|Z_1| \times |Z_2|$ solutions, all of which contain now n_3. This property of the search is the basis for the proof of the following theorem.

Theorem 1 *The size of the solution set, Z, constructed by the blind-search algorithm in Alg. B.1, for an EST of height h and branching factor b, is given by the* $(h-1)$th *term of the*

Input: The node n.

Output: The set Z of solutions for pruning all the branches of the tree.

blind-search(n) **:**

$N \leftarrow$ **generate-children**(n)
IF $N = \{\}$: RETURN $\{\}$
$N \leftarrow$ **sort-by-cost**$^a(N)$
$Z \leftarrow \{\{$**head**$(N)\}\}$
FOR $i = 1$ TO **length**(N)
    $Z_1 \leftarrow$ **blind-search**(n_i), $n_i \in N$
    $\forall z_j \in Z_1$: $Z \leftarrow Z \bigcup$ **replace-node**$^b(n_i, Z, z_j, n_{i+1})$
    $Z \leftarrow Z \bigcup Z_1$
ENDFOR
RETURN Z

[a] The children of a node are sorted in descending cost order.
[b] It replaces n_i, with z_j and n_{i+1}, in all members of Z.

Algorithm B.1: The blind-search algorithm, constructing the set of all possible pruning solutions for distance specialisation. Z_1 is a set of solutions, z_j an individual solution and n_i a node, which also serves as a pruning choice.

sequence:

$$\begin{aligned} S_1 &= 1 \\ S_2 &= b+1 \\ S_n &= \frac{1 - S_{n-1}^{(b+1)}}{1 - S_{n-1}}. \end{aligned}$$

Proof 1 *Using this recursive algorithm, the construction of the solution set Z for a tree of height h is achieved by the construction of equivalent sets for the b subtrees of height $h-1$. Each solution for one subtree, provides further variants of all the solutions in all previous subtrees, with the use of* **replace-node**. *Thus, the size of Z can be calculated by the following recursive function:*

$$\begin{aligned} \textbf{sizeof}(Z) &= 1 + \textbf{sizeof}(Z_1) + \textbf{sizeof}(Z_1) \times \textbf{sizeof}(Z_2) + \ldots + \textstyle\prod_{i=1}^{b} \textbf{sizeof}(Z_i), \\ \textbf{sizeof}(\{n\}) &= 1, \end{aligned}$$

where Z_i are the solution subsets as generated by **blind-search**. *The first term in the recursive function corresponds to pruning the root, then the solutions Z_1 for the most expensive child are calculated, then for the next most expensive, Z_2, and each one is included in the members of Z_1, and so on. However, the height and branching factor of all subtrees is the same and therefore it holds that:*

$$\textbf{sizeof}(Z_1) = \textbf{sizeof}(Z_2) = \ldots = \textbf{sizeof}(Z_b).$$

Consequently:

$$\begin{aligned} \textbf{sizeof}(Z) &= 1 + \textbf{sizeof}(Z_1) + \textbf{sizeof}(Z_1)^2 + \ldots + \textbf{sizeof}(Z_1)^b \\ &= \begin{cases} (b+1), \text{if } \textbf{sizeof}(Z_1) = 1 \text{ or} \\ \frac{1-\textbf{sizeof}(Z_1)^{(b+1)}}{1-\textbf{sizeof}(Z_1)}, \text{otherwise.} \end{cases} \end{aligned}$$

Moreover, it is clear that the size of the solution set is 1 *only for* $h = 2$*: the solution then is to prune the most expensive child. Therefore, the result of the above recursive function is equivalent to the* $(h-1)$th *term of the sequence presented in theorem 1.*

B.2 Specialisation of distance and duration

The search space for the specialisation algorithm increases when the choice of pruning by duration is added. The reason for this is that the number of pruning solutions increases. Moreover there is an important difference between the two parameters: sorting the children of a node by duration results in a different order than sorting them by distance. When using one of the two, the recursive blind-search algorithm of Alg. B.1 generates the complete solution set. This ceases to be true, when the two parameters are combined. In the following discussion the assumption is made that the children of a node are sorted in ascending order of distance from their parent (i.e., decreasing specialisation cost by distance) and ascending order of duration cost. This is a worst-case assumption. In addition, the assumption of single-sided specialisation holds as in the previous section. Algorithm B.2 constructs the complete solution set.

Input: The node n.

Output: The set Z of solutions for pruning all the branches of the tree.

blind-search(n) **:**

$N \leftarrow$ **generate-children**(n)
$N \leftarrow$ **sort-by-cost**(N)
$Z \leftarrow$ **explicit-pruning**[a](N)
FOR $i = 1$ TO **length**(N)
    $Z_1 \leftarrow$ **blind-search**(n_i), $n_i \in N$
    $\forall z_j \in Z_1$: $Z \leftarrow Z \bigcup$ **replace-node**[b](n_i, Z, z_j)
    $Z \leftarrow Z \bigcup Z_1$
ENDFOR
RETURN Z

[a] All the combinations of explicitly pruning the children of n.
[b] Replace n_i, in all members of Z, where it is used for pruning.

Algorithm B.2: The blind search algorithm, for specialising with distance and duration. Z_i, z_i and n_i are used as in Alg. B.1.

Although, in principle this algorithm is the same as the one in Alg. B.1, it differs in the following important points:

- As was seen in chapter 4, section 4.5.2, there are more than one ways of pruning a set of siblings explicitly. If $N = \{n_1, n_2, \ldots, n_b\}$ is the set of children of n and $d(n_i)$ and $s(n_i)$ are used to represent pruning n_i by duration and by distance respectively, the set Z_1 of solutions pruning a set of siblings is:

  $$Z_1 = \{\{s(n_1)\}, \{s(n_2), d(n_1)\}, \{s(n_3), d(n_2)\}, \ldots, \{s(n_b), d(n_{b-1})\}, \{d(n_b)\}\},$$

  since $s(n_i)$ causes the implicit pruning of $\{n_{i+1}, n_{i+2}, \ldots, n_b\}$, and $d(n_i)$ prunes implicitly $\{n_1, n_2, \ldots, n_{i-1}\}$. This result is due to the worst-case assumption that duration costs are ordered in exactly the opposite way than distance costs. As a result there are $(b+1)$ solutions for $h = 2$. This is equivalent to the worst-case scenario, i.e., exhaustive enumeration, for the ORAsib algorithm, Alg. 4.4.

- The **replace-node** function is different. Since there are now more than one ways of pruning the siblings of a node n_i, each of the pruning solutions for the children of n_i must be added to each of the alternative solutions for pruning the siblings of n_i. In Alg. B.1 there was only one such solution, pruning n_{i+1}.
- The root node of the tree can also be explicitly pruned by duration. The result of this is one additional solution for the whole tree.

In order to show how this algorithm constructs the solution set and illustrate the complexity of the search space, the simple example tree presented in Fig. B.1 will be used. This tree has $h = 3$ and $b = 3$. The number of solutions explicitly pruning all children of the root of the tree is therefore $u = (b + 1) = 4$, i.e., the following set:

$$\{\text{imp}(n_0) = \{\{s(n_3)\}, \{s(n_2), d(n_3)\}, \{s(n_1), d(n_2)\}, \{d(n_1)\}\}.$$

Note that the assumption made here is that the rightmost node in a set of siblings has the highest distance-pruning cost, similar to the ESTs seen so far. As a result, the leftmost sibling has the highest duration-pruning cost. Note also that the set of solutions, corresponding to the explicit pruning of the children of node n_0 is denoted by imp(n_0). Actually, the following is true for all non-leaf nodes:

$$|\text{imp}(n_0)| = |\text{imp}(n_1)| = |\text{imp}(n_2)| = |\text{imp}(n_3)| = 4 = u.$$

For this reason u will be used as the basic cost unit.

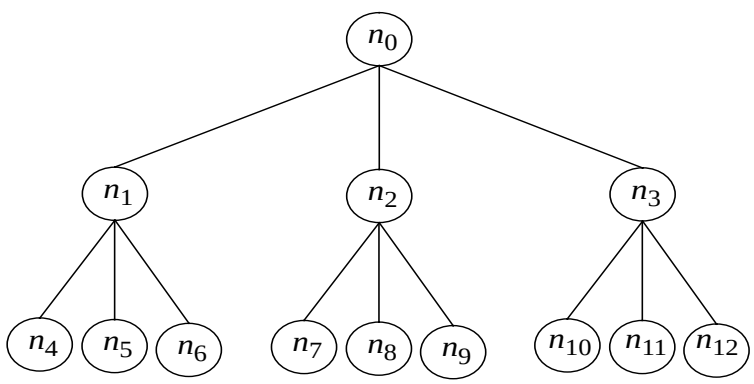


Figure B.1: A simple arbitrary EST.

Figure B.2 presents a sketch of the recursive process generating the complete set of solutions. In the first step, the set imp(n_0) is generated as described above. As a technical trick to aid the explanation, all the siblings which are implicitly pruned by duration are shown as being pruned explicitly by duration. Thus, the first set of solutions becomes:

$$\{\text{imp}(n_0) = \{\{s(n_3)\}, \{s(n_2), d(n_3)\}, \{s(n_1), d(n_2), d(n_3)\}, \{d(n_1), d(n_2), d(n_3)\}\}.$$

The reason for doing this is that the subset of explicit pruning solutions for the set of siblings $\{n_1, n_2\}$, i.e., ignoring n_3, can easily be constructed by removing $\{s(n_3)\}$ and $d(n_3)$ from the remaining solutions. Thus the possible combinations for pruning $\{n_1, n_2\}$ explicitly is:

$$\{\{s(n_2)\}, \{s(n_1), d(n_2)\}, \{d(n_1), d(n_2)\}\}.$$

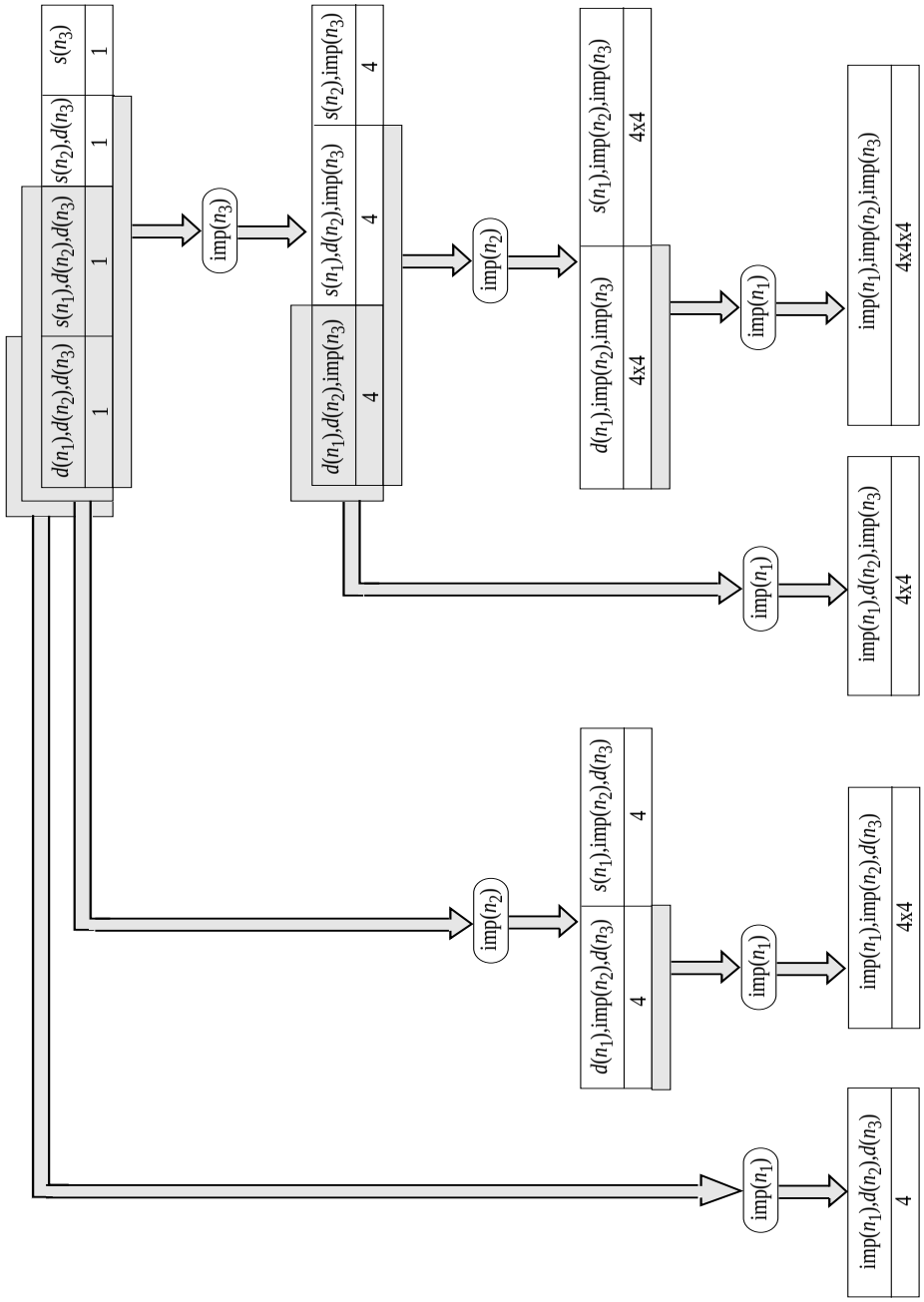


Figure B.2: Sketch of the recursive solution construction process. imp(n_i) stands for the set of pruning solutions for the children of n_i. The numbers below the solution sets correspond to their size.

Using this trick, at each recursive step, a new set of solutions is generated, by replacing $d(n_i)$, with its imp(n_i), i.e., the set of explicit pruning solutions for the children of n_i. This set is not explicitly shown in Fig. B.2, but it corresponds to $u = 4$ different solutions. The number of solutions in each solution set is shown below the set. The arrows show the replacement of $d(n_i)$ by imp(n_i).

Concentrating on the number of solutions, rather than the actual solution set, the sketch of Fig. B.2 can be translated into the tree of Fig. B.3, which reveals an interesting recursive pattern in the arrangement of sizes of solution sets.

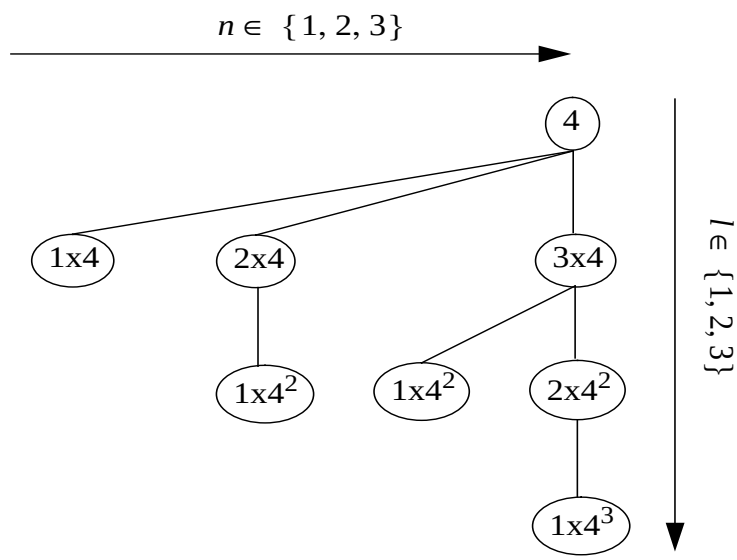


Figure B.3: A tree showing the recursive pattern in solution set sizes. l corresponds to the power to which the basic unit $u = 4$ is raised. n is the multiplicative term in each node.

Excluding the root of the tree, this pattern is expressed by the recursive function (B.1), which models the behaviour of Alg. B.2.

$$\begin{aligned} f(l,n) &= \sum_{i=0}^{n-1}(n-i)u^l + \sum_{i=1}^{n-1} f(l+1, n-i),\ l,n \in \mathbb{N}^+ \text{ and} \\ f(l,1) &= u^l,\ l \in \mathbb{N}^+, \end{aligned} \tag{B.1}$$

where l and n increase as shown in Fig. B.3. Combining this with the size on the root of the tree in Fig. B.3, the size of the search space is:

$$\textit{sizeof}(Z) = u + f(1, b).$$

The recursive function (B.1) can be simplified as follows. The tree of Fig. B.3 can be translated into the table B.1, calculating the number of identical nodes in the tree, with variable l and n. Concentrating then on the information that is not directly dependent on l and n, the third dimension of the table can be extracted as shown in table B.2. In order to make more obvious

the pattern of interest, a similar table for $h = 3$ and $b = 4$ is presented in Tab. B.3. The pattern emerging is Pascal's triangle and the size of the search space can be calculated using binomial expansion. The alternative function is:

$$g(n) = \sum_{i=1}^{n} iu(1+u)^{n-i}, n \in \mathbb{N}^{+} \tag{B.2}$$

and the size of the solution set is calculated by $g(b)$.

$$\textit{sizeof}(Z) = u + g(b).$$

In other words, $g(b) = f(1, b)$, which is proven below. The binomial function in (B.2) avoids the recursion performed in (B.1), simplifying the calculation of the solution.

l	n		
	1	2	3
1	$1 \times 1 \times 4$	$1 \times 2 \times 4$	$1 \times 3 \times 4$
2	$2 \times 1 \times 4^2$	$1 \times 2 \times 4^2$	—
3	$1 \times 1 \times 4^3$	—	—

Table B.1: Tabular presentation of the tree of Fig. B.3.

l	n		
	1	2	3
1	1	1	1
2	2	1	—
3	1	—	—

Table B.2: The third dimension of table B.1.

l	n			
	1	2	3	4
1	1	1	1	1
2	3	2	1	—
3	3	1	—	—
4	1	—	—	—

Table B.3: Equivalent table to B.2 for $h = 4$, showing Pascal's triangle pattern .

Theorem 2 *Given the two functions in (B.1) and (B.2), the following correspondence holds:*

$$f(1, b) = g(b). \tag{B.3}$$

Proof 2

Step 1. *Let the recursive function g_1 be defined as:*

$$\begin{aligned} g_1(l, n) &= \sum_{i=1}^{n} (iu^l + g_1(l+1, n-i)),\ l, n \in \mathbb{N}^{+} \text{ and} \\ g_1(l, 0) &= 0,\ l \in \mathbb{N}^{+}. \end{aligned} \tag{B.4}$$

Using this function, (B.3) can be proven if:

$$g_1 = f \tag{B.5}$$

and

$$g(b) = g_1(1, b). \tag{B.6}$$

Step 2. (B.4) *can be rewritten as follows:*

$$\begin{aligned} g_1(l,n) = & \sum_{i=1}^{n}(iu^l + g_1(l+1, n-i)) = \\ & \sum_{i=1}^{n} iu^l + \sum_{i=1}^{n} g_1(l+1, n-i) = \\ & \sum_{i=0}^{n}(n-i)u^l + g_1(l+1, 0) + \sum_{i=1}^{n-1} g_1(l+1, n-i) = \\ & \sum_{i=0}^{n-1}(n-i)u^l + \sum_{i=1}^{n-1} g_1(l+1, n-i). \end{aligned}$$

Also: $g_1(l,1) = \sum_{i=1}^{1}(iu^l + g_1(l+1, 1-i)) = u^l + 0 = u^l$. *It is clear, therefore, that* $g_1 = f$.

Step 3. *In* **steps 4-8** *the following relationship is proven:*

$$\forall n \in \mathbb{N}^+, g(n) = \sum_{i=1}^{n}(iu + ug(n-i)), \tag{B.7}$$

while it is clear that $g(0) = 0$. *This recursive form of* g, *allows the correspondence between* g *and* g_1 *as follows:*

$$g_1(l,n) = u^{(l-1)} \times g(n)$$

and therefore:

$$g_1(1,b) = \sum_{i=1}^{b}(iu + g_1(2, b-i)) = \sum_{i=1}^{b}(iu + ug(b-i)) = g(b),$$

proving (B.6) and (B.3).

Step 4. (B.2) *can be rewritten as follows:*

$$\begin{aligned} & g(n) = \\ & \sum_{i=1}^{n} iu[1 + \binom{n-i}{n-(i+1)}u^1 + \binom{n-i}{n-(i+2)}u^2 + \ldots + \binom{n-i}{1}u^{n-(i+1)} + \binom{n-i}{0}u^{n-i}] = \\ & \sum_{i=1}^{n} iu + \sum_{i=1}^{n} iu^2[\binom{n-i}{n-(i+1)} + \binom{n-i}{n-(i+2)}u^1 + \ldots + \binom{n-i}{1}u^{n-(i+2)} + \binom{n-i}{0}u^{n-(i+1)}] = \\ & \sum_{i=1}^{n} iu + \quad u^2\Big[1 \times [\binom{n-1}{n-2} + \binom{n-1}{n-3}u^1 + \ldots + \binom{n-1}{1}u^{n-3} + \binom{n-1}{0}u^{n-2}] + \\ & \qquad 2 \times [\binom{n-2}{n-3} + \binom{n-2}{n-4}u^1 + \ldots + \binom{n-2}{1}u^{n-4} + \binom{n-2}{0}u^{n-3}] + \\ & \qquad \vdots \\ & \qquad (n-2) \times [\binom{2}{1} + \binom{2}{0}u^1] + \\ & \qquad (n-1) \times [\binom{1}{0}]\Big]. \end{aligned} \tag{B.8}$$

Step 5. *Similarly,* (B.7) *can be expanded:*

$$
\begin{aligned}
&g(n) = \\
&\sum_{i=1}^{n} iu + \sum_{i=1}^{n} u^2 \sum_{j=1}^{n-i} j(1+u)^{n-(j+i)} = \\
&\sum_{i=1}^{n} iu + u^2[\sum_{j=1}^{n-1} j(1+u)^{n-(j+1)} + \sum_{j=1}^{n-2} j(1+u)^{n-(j+2)} + \ldots + \sum_{j=1}^{2} j(1+u)^{1} + \sum_{j=1}^{1} j] = \\
&\sum_{i=1}^{n} iu + \sum_{i=1}^{n} u^2 \Big[[(1+u)^{n-2} + 2(1+u)^{n-3} + \ldots + (n-2)(1+u) + (n-1)] + \\
&\qquad [(1+u)^{n-3} + 2(1+u)^{n-4} + \ldots + (n-3)(1+u) + (n-3)] + \\
&\qquad \vdots \\
&\qquad [(1+u) + 2] + \\
&\qquad [1]\Big] = \\
&\sum_{i=1}^{n} iu + u^2 \Big[1 \times [(1+u)^{n-2} + (1+u)^{n-3} + \ldots + (1+u) + 1] + \\
&\qquad 2 \times [(1+u)^{n-3} + (1+u)^{n-4} + \ldots + (1+u) + 1] + \\
&\qquad \vdots \\
&\qquad (n-2) \times [(1+u) + 1] + \\
&\qquad (n-1) \times [1]\Big].
\end{aligned}
\tag{B.9}
$$

Step 6. *However, each of the terms in the expanded version of* (B.9) *can be translated as follows:*

$$
\begin{aligned}
&1 + (1+u) + \ldots + (1+u)^{n-(x+1)} + (1+u)^{n-x} = \\
&[\binom{0}{0}] + \\
&[\binom{1}{1} + \binom{1}{0}u] + \\
&\vdots \\
&[\binom{n-(x+1)}{n-(x+1)} + \binom{n-(x+1)}{n-(x+2)}u + \ldots + \binom{n-(x+1)}{1}u^{n-(x+2)} + \binom{n-(x+1)}{0}u^{n-(x+1)}] + \\
&[\binom{n-x}{n-x} + \binom{n-x}{n-(x+1)}u + \ldots + \binom{n-x}{1}u^{n-(x+1)} + \binom{n-x}{0}u^{n-x}] =
\end{aligned}
$$

$$
\begin{aligned}
&[\binom{0}{0} + \binom{1}{1} + \ldots + \binom{n-x}{n-x}] + \\
&[\binom{1}{0} + \binom{2}{1} + \ldots + \binom{n-x}{n-(x+1)}]u + \\
&\vdots \\
&[\binom{n-(x+1)}{0} + \binom{n-x}{1}]u^{n-(x+1)} + \\
&[\binom{n-x}{0}]u^{n-x} = \\
&\binom{n-(x-1)}{n-x} + \binom{n-(x-1)}{n-(x+1)}u + \ldots + \binom{n-(x-1)}{1}u^{n-(x+1)} + \binom{n-(x-1)}{0}u^{n-x}.
\end{aligned}
\tag{B.10}
$$

Step 7. *Therefore, using* (B.10), *equation* (B.9) *becomes:*

$$
\begin{aligned}
&g(n) = \\
&\sum_{i=1}^{n} iu + u^2\Big[1 \times [\binom{n-1}{n-2} + \binom{n-1}{n-3}u + \ldots + \binom{n-1}{1}u^{n-3} + \binom{n-1}{0}u^{n-2}] + \\
&\qquad 2 \times [\binom{n-2}{n-3} + \binom{n-2}{n-4}u + \ldots + \binom{n-2}{1}u^{n-4} + \binom{n-2}{0}u^{n-3}] + \\
&\qquad \vdots \\
&\qquad (n-2) \times [\binom{2}{1} + \binom{2}{0}u] + \\
&\qquad (n-1) \times [\binom{1}{0}]\Big].
\end{aligned}
\tag{B.11}
$$

Step 8. (B.8) *and* (B.11) *are identical and therefore (B.7) holds.*

The discussion so far has assumed that $h = 3$. Due to the recursive process, by which the solution set is generated, the extension of the proof to $h > 3$ involves simply the substitution of the solution set size at $(h-1)$ for the basic size unit u in (B.2). Therefore, the size at h is given by the hth term of the sequence:

$$
\begin{aligned}
S_1 &= 1 \\
S_2 &= b + 1 \\
S_n &= \sum_{i=1}^{n} iS_{n-1}(1 + S_{n-1})^{n-i}.
\end{aligned}
$$

Note that S_1 has to be added to the size calculated in this way.

Appendix C

Optimality of incremental specialisation

C.1 Distance-only specialisation

The distance-only incremental specialisation algorithm performs a search for the optimal set of distance-pruning choices for an EST. The optimality criterion is the specialisation cost function. The algorithm uses the exclusion dependencies between the nodes on the same level of the tree to derive the optimal solution without, in average, exhaustive enumeration of all possible solutions. In this section, the optimality of the solution generated by ORAspec – if one is generated – is proven. In particular the following two conditions should be satisfied:

1. The set of pruning choices that is found by the algorithm is always a legal solution.
2. This is the optimal solution.

Assume an EST $T = (N, E)$, where N the set of nodes and E the edges of the tree. The elements of E are denoted by the predicate **child**(n_i, n_j), where $n_i, n_j \in N$ and n_i is the child of n_j. Using the **child** predicate, the obvious predicates can be defined: **leaf**(n_i), **ancestor**(n_j, n_i), **parent**(n_j, n_i), **sibling**(n_i, n_j). Also the following concepts are used:

- **cousin**(n_i, n_j) is used for nodes at the same horizontal level of the EST.
- *Non-excludable* nodes are nodes which cannot be pruned explicitly, i.e., the corresponding distance value between them and their parent satisfies the LGG_s model.
- c_s is the cost of specialisation as defined in chapter 4 for a node and a set of nodes, representing the equivalent pruning choices.

Definition 2 *Exclusion. A node n_i excludes a node n_j if they are cousins and the cost of the former is at least as high as that of the latter:*

$$n_i >_e n_j \text{ iff. } \textbf{cousin}(n_i, n_j) \wedge c_s(n_i) \geq c_s(n_j).$$

Definition 3 *Solution. A set S of nodes of the EST T is called a solution if and only if the following hold:*

$$\forall n_i \in T, \textbf{leaf}(n_i) : \begin{cases} n_i \in S, \text{ or} \\ \exists n_j \in S : n_j >_e n_i \text{ or} \\ \exists n_j \in S : \textbf{ancestor}(n_j, n_i) \text{ or} \\ \exists n_j, n_k \in T : n_j \in S \wedge \textbf{ancestor}(n_k, n_i) \wedge n_j >_e n_k. \end{cases}$$

and $\nexists n_j, n_k \in S, n_j >_e n_k$. In other words, each leaf node must:

- *either be in the solution,*
- *or be excluded by a node in the solution,*
- *or have an ancestor in the solution,*
- *or have an ancestor, which is excluded by a node in the solution.*

Note that this definition does not guarantee that a solution is a legal one, because it may include non-excludable nodes. However, it guarantees that all leaf nodes are covered and there are no nodes in the solution which are excluded by other nodes in the same solution. A solution S will be called **partial** *if it includes non-excludable nodes and* **complete** *if it does not.*

The principle underlying the ORAspec algorithm is that if the search starts from a solution, i.e., a set of nodes satisfying definition 3, then a new solution can be generated by replacing a non-leaf node $n_i \in S$ with:

a. the maximum-cost node n_j, among the cousins of n_i, in the same specialisation set, i.e., left or right, and
b. the maximum-cost left- and right-pruning children, (n_l, n_r), of n_i and
c. all non-excludable children Q_i of n_i.

Also all the nodes which are excluded by other nodes in the solution are removed. Note that these can be either one of the n_l, n_r or any nodes that they subsume.

Definition 4 *Extension. A set of nodes S' is called an extension of a solution S, if it is constructed from S in the above way or it is constructed by applying this process to an extension of S.*

Theorem 3 *An extension S' of a solution S is itself a solution.*

Proof 3 *Assume that the node $n_i \in S$ is the one replaced in S', by n_j, n_l, n_r, Q_i. n_i is responsible for excluding two sets of leaves of the tree T:*

1. $F_1, \forall n_h \in F_1,$ **leaf**$(n_h) \wedge$ **ancestor**(n_i, n_h) *and*
2. $F_2, \forall n_h \in F_2, \exists n_k \in T,$ **ancestor**$(n_k, n_h) \wedge n_i >_e n_k$.

Assume that L_i, Q_i, R_i are the three sets of children of n_i. Q_i are explicitly included in the new set of nodes. Moreover: $\forall n_h \in L_i, n_l >_e n_h$ and $\forall n_h \in R_i, n_r >_e n_h$. Thus, it is trivially proven that all leaves in F_1 are still excluded, since either themselves or an ancestor of them is in L_i, Q_i, R_i. The leaves in F_2 are excluded by the cousin of n_i, n_j. Since n_j is the most expensive cousin of n_i, it holds that:

$$\forall n_h \in T, \textbf{cousin}(n_h, n_i) \wedge n_i >_e n_h \wedge n_i \neq n_h : \textbf{cousin}(n_h, n_j) \wedge n_j >_e n_h.$$

Therefore all nodes which were excluded by n_i are now excluded by n_j.

Corollary 1 *Starting with a solution S, which uses only children of the root of an EST, it follows from definition 4, that all solutions are extensions of it. One just needs to carry on applying the extension process to the generated solutions, until all branches of the tree have been traversed.*

Thus, the algorithm is guaranteed to generate sets of nodes, which are solutions. The legality of the final solution is also guaranteed, by an explicit check, on the final solution. If it includes non-excludable nodes it is rejected as a solution.[1] Thus, the first criterion, set at the beginning of the section is satisfied.

[1]This check happens every time a tree branch has been traversed to the leaves. In this way, if a sequence has been found, which contains only non-excludable nodes, the search stops and the empty set is returned.

The second optimality criterion can be proven by examining the properties of the termination condition for ORAspec. The algorithm stops, when the least-cost solution is complete and its upper and lower cost estimates are the same. If the latter is not satisfied, the most expensive node is expanded, i.e., it is replaced to produce an extension of the solution.

Definition 5 *Expanded node. Given a solution S, a node $n_i \in S$, is called* expanded *if it has been used to generate an extension to a solution in which it participates.*

Definition 6 *Cost bounds. The upper bound of a solution S is defined as the sum of the costs of all nodes of S:*

$$\textbf{high}(S) = \sum_{i=1}^{|S|} c_s(n_i), n_i \in S.$$

This is the true cost of the solution. The lower bound is the sum of the costs of expanded nodes in S.

$$\textbf{low}(S) = \sum_{j=1}^{|S|} c_s(n_j),\ (n_j \in S \wedge \textbf{expanded}(n_j)).$$

Definition 7 *Final solution. A solution S is called* **final** *if and only if it is* **complete** *and its upper cost bound equals its lower cost bound:*

$$\textbf{final}(S) \text{ iff. } \textbf{complete}(S) \wedge \textbf{low}(S) = \textbf{high}(S).$$

In other words, all of the nodes in S must be expanded.

Lemma 1 *An implication of definition 7 is that any extension S' of a solution S, which is a* **final** *solution, has a higher or equal low cost bound to S (i.e.,* $\textbf{low}(S') \geq \textbf{low}(S)$*). The reason for this is that in order to achieve a lower cost than* $\textbf{low}(S)$*, at least one expanded node $n_h \in S$ needs to be excluded by one other node $n_j \in S'$. This implies that $c_s(n_j) \geq c_s(n_h)$ and since all nodes in S' are expanded the low cost bound of S' must be higher than that of S.*

The ORAspec search terminates when the least-cost solution is final. In order for this to be the optimal solution there must not be an alternative one with a lower cost.

Theorem 4 *If a set of solutions $SSet$ is generated and the least-cost solution S_i in $SSet$ is* **final***, there is no other set of nodes S_j in the EST T, such that S_j constitutes a complete solution and the cost of S_j is lower than that of S_i:*

$$\nexists S_j \subset T, \textbf{complete}(S_j) \wedge\ (\textbf{high}(S_j) < \textbf{low}(S_i)).$$

Proof 4 *Assume that such a solution S_j exists. It holds that:*

1. *$S_j \notin SSet$, by the definition 7.*
2. *Also S_j cannot be an extension of S_i, by the lemma 1.*
3. *Due to corollary 1, there must be a solution S_h in $SSet$ of which S_j is an extension. However, it is known that* $\textbf{low}(S_h) \geq \textbf{low}(S_i)$*, $\forall S_h \in SSet$. Moreover, due to lemma 1, it is known that* $\textbf{low}(S_h) \leq \textbf{low}(S_j)$*, as S_j is an extension of S_h. Therefore* $\textbf{low}(S_i) \leq \textbf{low}(S_j) \leq \textbf{high}(S_j)$*, contradicting the initial assumption.*

C.2 Duration and distance specialisation

The extended version of the ORAspec algorithm differs from the distance-only version in one way: the optimal pruning choice for a set of cousin nodes N is not immediately obvious and requires search. This search is performed by the ORAsib algorithm, Alg. 4.4, which splits N into the independent subsets S and D, and searches for the optimal choice within each subset. The underlying claim is that if the optimal combination of pruning choices does not comprise of pruning the two most expensive nodes in S and D by their preferred parameter, then it can be found by examining each of the two sets individually.

The important property of the problem is the independence between S and D, which can be stated as follows:

Definition 8 *Independent cousins. None of the nodes in S and D are excluded by another node in S or D:*

$$\forall n_i, n_j \in S \bigcup D : \quad (c_d(n_i) > c_d(n_j)) \wedge (c_s(n_i) \leq c_s(n_j)) \vee (c_d(n_j) > c_d(n_i)) \wedge (c_s(n_j) \leq c_s(n_i)),$$

where c_d and c_s are the costs of pruning a node by duration and by distance respectively.

Lemma 2 *From definition 8 and the definition of S and D, which states that:*

$$\forall n_i \in S, n_j \in D : (c_s(n_i) \leq c_d(n_i)) \wedge (c_d(n_j) \leq c_s(n_j)),$$

it follows that:

$$\forall n_i \in S, n_j \in D : (c_s(n_i) < c_s(n_j)) \wedge (c_d(n_j) < c_d(n_i)), \tag{C.1}$$

Theorem 5 *Assuming $a \in S$, $b \in D$, such that:*

$$\forall n_i \in S, n_j \in D : c_s(n_i) \leq c_s(a) \wedge c_d(n_j) \leq c_d(b), \tag{C.2}$$

there does not exist a different pair of nodes $e \in S$, $f \in D$, such that:

$$\forall n_i \in S \bigcup D : (c_s(n_i) \leq c_s(e)) \vee (c_d(n_i) \leq c_d(f)) \quad \wedge \quad (c_s(e) + c_d(f)) < (c_s(a) + c_d(b)) \tag{C.3}$$

$$\vee$$

$$\forall n_i \in S \bigcup D : (c_d(n_i) \leq c_d(e)) \vee (c_s(n_i) \leq c_s(f)) \quad \wedge \quad (c_d(e) + c_s(f)) < (c_s(a) + c_d(b)) \tag{C.4}$$

Proof 5 *Suppose (C.3) is true. From (C.2) it follows that:*

$$(c_s(e) \leq c_s(a)) \wedge (c_d(f) \leq c_d(b)).$$

If

$$(c_s(e) = c_s(a) \wedge c_d(f) = c_d(b)),$$

then the second condition in (C.3) does not hold. Otherwise,

$$(c_s(e) < c_s(a) \vee c_d(f) < c_d(b)).$$

Moreover, from (C.1) it follows that:

$$\forall n_i \in S, n_j \in D : (c_s(n_i) < c_s(b)) \wedge (c_d(n_j) < c_d(a)) \rightarrow$$

$$(c_s(e) < c_s(b)) \wedge (c_d(f) < c_d(a))$$

Therefore, the first condition of (C.3) does not hold.

Suppose (C.4) is true. Again from (C.1) it follows that:

$$\begin{aligned} \forall n_i \in S, n_j \in D : \quad & (c_s(n_i) < c_s(f)) \wedge (c_d(n_j) < c_d(e)) \rightarrow \\ & (c_s(a) < c_s(f)) \wedge (c_d(b) < c_d(e)) \rightarrow \\ & (c_d(e) + c_s(f)) > (c_s(a) + c_d(b)) \end{aligned}$$

which contradicts the second part of (C.4).

Appendix D

Algorithms for full supervision

D.1 Building and updating a relative event support tree

This section describes the algorithms which incrementally construct a REST as the input stream of event occurrences is being processed and feedback is received from the training data. The first algorithm, Alg. D.1, initiates the updating process. It receives as input the terminal subevent and the event occurrence which should be recognised. In the general case, the begin point, B, of the terminal subevent occurrence is a set of points, because the begin point of an event occurrence in the feedback data is allowed to be undefined, i.e., $=?$. B will be a singleton, if e_i is an input event or the begin point for this occurrence of e_i, is defined in the feedback data, i.e., $\neq?$. The algorithm starts, **satisfy-duration**, by removing from B the points which do not satisfy the duration window, w_{id}, of e_i in e_s. Then the remaining set of begin points, B', is split, **split-by-begin**, into B^+ and B^-, according to whether they happen after or before the begin point of the supported event. B^+ is used for positive examples, if e_s should be recognised, and B^- always for negative. Then the duration, corresponding to each element of B^+ and B^- is calculated, **get-duration**, and the procedure for inserting the new node in the REST, **insert-node**, is called. Finally, the example counter is incremented, if the REST is indeed updated. The REST will not be updated, if no complete supporting sequence – either positive or negative – can be build.

insert-node, Alg. D.2, is a recursive procedure,[1] which builds the complete subtree of the newly added node. The following is a syntactic description of the REST:

```
<REST> ::= <Node>, <REST>
<REST> ::= <Node>
<Node> ::= <Parameters>, <Examples>, <REST>
<Parameters> ::= distance, duration
<Examples> ::= <Positive>, <Negative>
<Positive> ::= example
<Positive> ::= example, <Positive>
<Negative> ::= example
<Negative> ::= example, <Negative>
```

where the terminal symbols `distance` and `duration` correspond to the `(distance, duration)` pair of the node and `example` is an example label. Note that `duration` can be a list of duration values. In that case, one node represents a set of siblings. This representation

[1] It is called from **add-child**.

is chosen in order to reduce the storage requirements for the REST. **insert-node** simply calls **add-children** which builds the subtree and adds the new node to the REST. If a node with the same `(distance,duration)` pair already exists, then this node is updated, rather than creating a new one.

Algorithm D.3 presents **add-children**, which deals with the children of a node. If the supported event is a repeating one, the preceding subevent occurrence is retrieved, **get-preceding**, and the satisfaction of the corresponding distance and duration windows is checked. If the windows are not satisfied, the branch is not grown any further.[2] The same happens if the length of the repeating sequence, given by the current depth in the REST, l, is greater than the maximum number of repetitions. When the growth of the branch stops, if the minimum number of repetitions has been reached, **add-leaf** is called to check whether the example is positive or not, i.e., whether the begin point of the supported event occurrence is satisfied. If the minimum number of repetitions has not been reached, *ctype*$= -1$ is returned, meaning that the node has no children. If the preceding subevent occurrence is not offending the windows and the maximum number of repetitions has not been reached, the branch is grown further by calling **add-child**, which recursively calls **insert-node**. When the minimum number of repetitions is reached, **add-leaf** is called for every subsequence, reflecting the fact that each repeating subsequence is a separate example. If the supported event is not a repeating one and the first subevent in the precedence sequence has not been already examined, all of the preceding subevent occurrences are collected by **satisfy-precedes** and the satisfaction of the parameter windows is checked for each one. If none of them satisfies the windows, the supporting sequence is discarded. Otherwise, **add-child** is called for each preceding subevent occurrence. When the start of the precedence sequence is reached, **add-leaf** is called to check whether the example is a positive or a negative one.

Algorithm D.4 describes **add-child**, which processes a node in a similar way as the top-level **update-REST** algorithm. Their only difference is the use of the **update-min-begin**, which updates the set of minimum begin points, taking into account the new occurrence. **add-leaf**, Alg. D.5, is called when a complete sequence of subevent occurrences has been processed. It checks, **satisfy-begin-point**, whether the begin point of the supported occurrence is satisfied and returns the appropriate example type. It also updates the occurrence in the database.

[2] An alternative approach would be to skip the offending subevent occurrence and use the one before that.

Input: terminal subevent occurrence: $e_i(B, f)$; supported event occurrence: $e_s(b_1, f)$; example type: $t = +$ or $-$; the REST: T_s.

Output: the updated REST: T_s.

Globals: example counter: xm; duration windows: W_d; TCN links: S.

update-REST$(e_i(B, f), e_s(b_1, f), t, T_s)$ **:**

$(a_i, e_i, e_s, ((?, (?, ?)), d_i) \in S \vee (a_i, e_i, e_s, ((a_i, s_i), d_i) \in S$ % terminal subevent or repeating
$(a_i, w_{id}) \in W_d$
$B' \leftarrow$ **satisfy-duration**$(w_{id}, \{(B, f)\})$
$(B^+, B^-) \leftarrow$**split-by-begin**(B, b_1, f)
$(D^+, D^-) \leftarrow$**get-duration**(B^+, B^-, f)
`IF` $D^+ = \{\}$: $T'_s \leftarrow T_s$
`ELSE`: $T'_s \leftarrow$**insert-node**$(t, 0, D^+, ?, a_i, (B^+, f), B^+, e_s(b_1, f), T_s)$
`IF` $D^- \neq \{\}$: $T'_s \leftarrow$**insert-node**$(-, 0, D^-, ?, a_i, (B^-, f), B^-, e_s(b_1, f), T'_s)$
`IF` $T'_s \neq T_s$:
    $xm \leftarrow xm + 1$
    $T_s \leftarrow T'_s$
`ENDIF`
`RETURN` T_s

Algorithm D.1: Algorithm for updating the REST.

Input: example type: $t = +$ or $-$; current depth in the REST: l; duration list: D; distance s; subevent: a_i; stamp of preceded event: (B, f); earliest begin points so far: B_2; supported event occurrence: $e_s(b_1, f)$; the REST: T_0.

Output: the updated REST: T_0.

Globals: example counter: xm.

insert-node$(t, l, D, s, a_i, (B, f), B_2, e_s(b_1, f), T_0)$ **:**

`IF` $\exists((s, D), (P, N), T_1) \in T_0$: % update existing node
    $(T_1$,*ctype*$) \leftarrow$**add-children**$(t, l, a_i, (B, f), B_2, e_s(b_1, f), T_1)$
    `IF` *ctype*$= 1$: $P \leftarrow P \bigcup \{xm\}$
    `ELSEIF` *ctype*$= 0$: $N \leftarrow N \bigcup \{xm\}$
    `ENDIF`
`ELSE`: % new node
    $(T_2$,*ctype*$) \leftarrow$**add-children**$(t, l, a_i, (B, f), B_2, e_s(b_1, f), \{\})$
    `IF` *ctype*$= 1$: $T_0 \leftarrow T_0 \bigcup \{((s, D), (\{\}, \{xm\}), T_2)\}$ % negative
    `ELSEIF` *ctype*$= 0$: $T_0 \leftarrow T_0 \bigcup \{((s, D), (\{xm\}, \{\}), T_2)\}$ % positive
    `ENDIF`
`ENDIF`
`RETURN` T_0

Algorithm D.2: Algorithm for updating a REST node or inserting a new one.

Input: example type: $t = +$ or $-$; current depth in the REST: l; subevent: a_i; stamp of preceded subevent occurrence: (B, f); earliest begin points so far: B_2; supported event occurrence: $e_s(b_1, f_0)$; the REST: T_0.

Output: the updated REST: T_0; type-of-children flag: *ctype*.

Globals: occurrence histories: $\mathcal{H}$; duration and distance windows: W_d and W_s; TCN links: S.

add-children$(t, l, a_i, (B, f), B_2, e_s(b_1, f_0), T_0)$ **:**

```
l ← l + 1
IF ∃(a_i, e_i, e_s, ((a_i, s_i), d_i)) ∈ S:                    % repeating
        repetitions(e_i, (n_i^-, n_i^+)), (e_i, H_i) ∈ H
        (a_i, w_id) ∈ W_d, (a_i, w_is) ∈ W_s
        H_i ← get-preceding^a(H_i, f)
        H_i ← satisfy-windows(H_i, B, w_id, w_is)
        IF (H_i = {} ∨ l > n_i^+):                             % stop branch growth
                IF l ≥ n_i^-: ctype←add-leaf(t, B_2, e_s(b_1, f))
                ELSE: ctype← −1
                RETURN (T_0,ctype)
        ELSE:                                                  % continue branch growth
                {(B_i, f_i)} ← H_i                             % singleton set
                s ← get-distance^b(f_i, B, f)
                IF l ≥ n_i^-:
                        ctype_1 ←add-leaf(t, B_2, e_s(b_1, f))
                        (T_0,ctype_2) ←add-child(t, l, s, a_i, (B_i, f_i), B_2, e_s(b_i, f_0), T_0)
                        ctype← max(ctype_1,ctype_2)
                ELSE: (T_0,ctype) ←add-child(t, l, s, a_i, (B_i, f_i), B_2, e_s(b_i, f_0), T_0)
                ENDIF
                RETURN (T_0,ctype)
        ENDIF
ELSEIF ∃(a_j, e_j, e_s, ((a_i, s_j), d_j)) ∈ S:                % non-leaf
        (e_j, H_j) ∈ H
        (a_j, w_jd) ∈ W_d, (a_j, w_js) ∈ W_s
        H_1 ←satisfy-precedes(H_j, f)
        H_1 ←satisfy-duration(w_jd, H_1), H_1 ←satisfy-distance(w_js, B, H_1)
        IF H_1 = {}: RETURN (T_0,ctype= −1)
        ELSE:
                ctype ← −1
                ∀(B_k, f_k) ∈ H_1:
                        s ← get-distance(f_k, B, f)
                        (T_0,ctype_1) ←add-child(t, l, s, a_j, (B_k, f_k), B_2, e_s(b_i, f_0), T_0)
                        ctype← max(ctype_1,ctype)
                RETURN (T_0, ctype)
        ENDIF
ELSE:                                                          % leaf node
        ctype←add-leaf(t, B_2, e_s(b_1, f))
        RETURN (T_0,ctype)
ENDIF
```

[a] It returns either one occurrence or an empty set.

[b] If B is singleton ($\{b\}$), it returns a number corresponding to the distance ($s = b - f_k$) of the preceding subevent occurrence from the preceded one, otherwise, it uses the end point of the preceded subevent ($s^* = f - f_k$) and returns $s = s^*$−d, where the character d signifies that s corresponds to a set of distances, that can be derived if the duration of the preceded event is known.

Algorithm D.3: Algorithm for updating recursively the children of a REST node.

Input: example type: $t = +$ or $-$; current depth in the REST: l; distance value for the new node: s; subevent: a_i; stamp of the subevent occurrence: (B, f); earliest begin points so far: B_2; supported event occurrence: $e_s(b_1, f_0)$; the REST: T_0.

Output: the updated REST: T_0; type-of-children flag: *ctype*.

add-child$(t, l, s, a_i, (B, f), B_2, e_s(b_1, f_0), T_0)$ **:**

$(B^+, B^-) \leftarrow$ **split-by-begin**(B, b_1, f)
$(D^+, D^-) \leftarrow$ **get-duration**(B^+, B^-, f)
$B_2^+ \leftarrow$**update-min-begin**(B^+, B_2)
`IF` $D^+ = \{\}$: $T^+ \leftarrow \{\}$
`ELSE`: $T^+ \leftarrow$**insert-node**$(t, l, D^+, s, a_j, (B^+, f), B_2^+, e_s(b_1, f_0), T_0)$
$B_2^- \leftarrow$**update-min-begin**(B^-, B_2)
`IF` $D^- = \{\}$: $T^- \leftarrow \{\}$
`ELSE`: $T^- \leftarrow$**insert-node**$(-, D^-, s, a_j, (B^-, f), B_2^-, e_s(b_1, f_0), T^+)$
`IF` $T^+ \neq T_0$: `RETURN` $(T_0 = T^-$,*ctype*$= 1)$ % positive children
`ELSEIF` $T^- \neq T_0$: `RETURN` $(T_0 = T^-$,*ctype*$= 0)$ % negative children only
`ELSE`: `RETURN` T_0, $ctype = -1$ % no children
`ENDIF`

Algorithm D.4: Algorithm for inserting each child node.

Input: example type: $t = +$ or $-$; earliest begin points so far: B_2; supported event occurrence: $e_s(b_1, f_0)$.

Output: the type-of-children flag: *ctype*.

add-leaf$(t, B_2, e_s(b_1, f_0))$ **:**

$B_2^+ \leftarrow$**satisfy-begin-point**(B_2, b_1)
$B_2^- \leftarrow B_2 \backslash B_2^+$
update-begin-point[a](e_s, f_0, B_2^-, B_2^+)
`IF` $(t = + \wedge B_2^+ \neq \{\})$: `RETURN` $ctype = 1$ % positive
`ELSE`: `RETURN` $ctype = 0$ %negative

[a]Updates the occurrence in the database, to store the alternative positive and negative begin points. They are needed to check the subsumption in repeating events.

Algorithm D.5: Process leaf node.

D.2 Local search algorithm

The local refinement algorithm generates and evaluates local solutions, as explained in section 5.4.1. The syntactic definition for a local solution is as follows:

```
<LSol> ::= subevent,<Ranges>,<Cover>,<Fitness>
<Ranges> ::= <SRange,DRange>
<SRange> ::= low,high
<DRange> ::= low,high
<Cover> ::= positive,negative
<Fitness> ::= proximity,purity
```

where `subevent` identifies the examined level of the REST, `low` and `high` the lower and upper bound of the range, `positive` and `negative` the number of positive and negative examples that are covered and `proximity` and `purity` the fitness score for the solution.

Algorithm D.6 generates and evaluates local solutions for a set of REST nodes. In general it performs a two-dimensional search. It generates all local solutions, **generate-local**, on the distance and duration axes separately, chooses the best local solution, **best-local**, and expands it on the opposite axis. The **fitness** function evaluates the local solutions. **LRAloc** returns a list of solutions, of which the head is the best local solution and the tail all others. Among the suboptimal solutions, there may be some which are only partially constrained, i.e., the result of one-dimensional search. If one of these is chosen and fed back to **LRAloc** only a one-dimensional search is needed. In that case, the range (r^-, r^+) provides the results of the previous search and R will be a singleton, containing the parameter that has not been examined yet. The same happens with all solutions for the terminal subevent, for which only the duration range is defined.

Algorithm D.7 describes the **generate-local** procedure, which generates local solutions, on one of the two dimensions of the feature space. The choice on the other dimension may have already been made, in which case, $(r^-, r^+) \neq (?, ?)$. The algorithm first skips all the REST nodes that do not cover any positives, up to the low limit of the original range. Note that the nodes are ordered by r and the skipping takes place along this dimension. Then the low limit for the next set of local solutions is established by **low-limit**, choosing between the parameter value of the first REST node and the value that **skip-negative** returns. If the examined event is a repeating one, the nodes in T_0 will hold distance and duration ranges, rather than single values. Once the low limit has been established, the low limit of each range in the REST nodes can be ignored and they can be treated as ordinary nodes. This is done by **translate-rep**. **all-high-values** generates all useful ranges, with the given low limit, and returns the corresponding local solutions. Then the REST nodes with the same parameter value as the first node of T_0 are skipped, **skip-same-value**, and **generate-local** is called recursively, to choose the next low limit and generate a new set of local solutions.

Algorithm D.8 presents **all-high-values** which generates all useful local solutions, with a common low limit. The aim of the algorithm is to skip as many of the candidate upper limits for local solutions as possible. It first skips all the REST nodes which do not cover positive examples and their parameter value, v_2, is greater than the upper limit of the original parameter range, v_h. This is done on the ground that their proximity and purity can only be lower than the last examined solution. They cover more negative examples and their upper limit is further from the upper limit of the original range.[3] Then all the nodes which do not cover negative examples are examined. None of these solutions are ignored. Once a node has been reached which covers negative examples, all nodes with the same parameter value are processed, without ignoring any solutions, and **all-high-values** is called recursively. The algorithm uses the procedures **make-local** and **insert-local**. The former one constructs a local solution from its components: subevent, parameter ranges, covered positive and negative examples. It also calculates the fitness of the solution, with the use of the original parameters (proximity) and the complete set of examples (purity). **insert-local** inserts a solution to the solution list as described in Alg. D.9.

Algorithm D.9, **insert-local**, is responsible for updating the solution list, when a new local solution is generated. It makes sure that old solutions subsumed by the new one are removed and maintains the set in descending fitness order. If the new solution is subsumed by an old one, it is not added to the set. A solution is subsumed by another if its ranges are subsumed, **subsumes**, it has a lower proximity and covers more negative examples than the other solution.

[3] An even more drastic reduction can be achieved by ignoring all nodes which cover only negative examples, except the one closest to the original range. Local solutions would then not be exhaustively enumerated.

Input: examined subevent: a_i; set of non-examined parameters: R; REST: T; selected range on the opposite dimension: (r^-, r^+); number of positive: p_0 and negative: n_0 examples covered so far.

Output: a set of generated local solutions: L.

Globals: TCN links: S.

LRAloc$(a_i, R, T, (r^-, r^+), p_0, n_0)$ **:**

$(a_i, e_i, e_s, ((a_j, s_i), d_i)) \in S$

`IF` $R = \{r\}$ % terminal subevent or partially constrained solution

  $T \leftarrow$ **sort-by**[a](r, T)

  `RETURN` $L =$**generate-local**$(a_i, r, ?, s_i, d_i, (r^-, r^+), p_0, n_0, T, \{\})$

`ELSE:` % otherwise a 2-d search is needed

  $T_1 \leftarrow$**sort-by**(s, T)

  $L_0 \leftarrow$**generate-local**$(a_i, \mathrm{s}, ?, s_i, d_i, (?, ?), p_0, n_0, T_1, \{\})$

  $T_2 \leftarrow$**sort-by**(d, T)

  $L_1 \leftarrow$**generate-local**$(a_i, \mathrm{d}, ?, s_i, d_i, (?, ?), p_0, n_0, T_2, \{\})$

  $(a_i, (s_1, d_1), p_1, n_1, (\mathit{prox}_1, \mathit{pur}_1)) \leftarrow$**best-local**$(L_0)$

  $L_2 \leftarrow L_0 \backslash \{(a_i, (s_1, d_1), (p_1, n_1), (\mathit{prox}_1, \mathit{pur}_0))\}$

  $(a_i, (s_2, d_2), (p_2, n_2), (\mathit{prox}_2, \mathit{pur}_1)) \leftarrow$**best-local**$(L_1)$

  $L_2 \leftarrow L_2 \bigcup L_1 \backslash \{(a_i, (s_2, d_2), (p_2, n_2), (\mathit{prox}_2, \mathit{pur}_2))\}$

  $f_1 \leftarrow$**fitness**$(\mathit{prox}_1, \mathit{pur}_1)$, $f_2 \leftarrow$**fitness**$(\mathit{prox}_1, \mathit{pur}_1)$

  `IF` $f_1 \geq f_2$:

    $T_1 \leftarrow$ **nodes-in-range**$(T_1, (s_1, d_1))$

    $L_2 \leftarrow L_2 \bigcup \{(a_i, (s_2, d_2), (p_2, n_2), (\mathit{prox}_1, \mathit{pur}_1))\}$

    $L_3 \leftarrow$ **generate-local**$(a_i, \mathrm{d}, ?, s_i, d_i, s_1, p_1, n_1, T_1, \{\})$

  `ELSE:`

    $T_2 \leftarrow$ **nodes-in-range**$(T_2, (s_2, d_2))$

    $L_2 \leftarrow L_2 \bigcup \{(a_i, (s_1, d_1), (p_1, n_1), (\mathit{prox}_1, \mathit{pur}_0))\}$

    $L_3 \leftarrow$ **generate-local**$(a_i, \mathrm{s}, ?, s_i, d_i, d_2, p_2, n_2, T_2, \{\})$

  `ENDIF`

  `RETURN` $L = L_2 \bigcup L_3$

`ENDIF`

[a] Sort the set of REST nodes by that parameter.

Algorithm D.6: Algorithm for generating and evaluating local solutions.

Input: subevent: a_i; examined parameter: r; the low limit for the generated ranges: v_l; original distance and duration ranges for the token: (s^-, s^+) and (d^-, d^+); selected range on the opposite dimension: (r^-, r^+); total number of positive: p and of negative: n examples in the REST; REST: T; local solutions generated so far: L.

Output: new set of local solutions: L.

generate-local$(a_i, r, v_l, (s^-, s^+), (d^-, d^+), (r^-, r^+), p, n, T, L)$ **:**

$(v_0, T_0) \leftarrow$**skip-negative**$(r, v_l, (s^-, s^+), (d^-, d^+), T)$
`IF` $T_0 = \{\}$: `RETURN` L
`ELSE:` % not all negative
  $v_1 \leftarrow$**low-limit**$(r,$**head**$(T_0), v_0)$
  $T_0 \leftarrow$**translate-rep**(r, T_0)
  $L \leftarrow$**all-high-values**$(a_i, r, v_1, (s^-, s^+), (d^-, d^+), (r^-, r^+), \{\}, 0, \{\}, p, n, T_0, L)$
  $((s, d), (P_0, N_0), T_1) \leftarrow$**head**$(T_0)$
  $v_1 \leftarrow$**parameter**[a]$(r, (s, d))$
  $(v_1, T_0) \leftarrow$**skip-same-value**(r, T_0)
  `IF` $T_0 = \{\}$: `RETURN` L
  `ELSE:` `RETURN` $L =$**generate-local**$(a_i, r, v_1, (s^-, s^+), (d^-, d^+), (r^-, r^+), p, n, T_0, L)$
`ENDIF`

[a] Returns the value of the examined parameter.

Algorithm D.7: Algorithm for generating local solutions on one dimension.

Input: subevent: a_i; examined parameter: r; the low limit for the generated ranges: v_l; original distance and duration ranges for the subevent: (s^-, s^+) and (d^-, d^+); selected range on the opposite dimension: (r^-, r^+); set of positive: P_1 and number of negative: n_1 examples covered so far; covered REST nodes: T_1; total number of positive: p and of negative: n examples for the REST; non-examined REST: T; local solutions generated so far: L.

Output: new set of local solutions: L.

all-high-values$(a_i, r, v_l, (s^-, s^+), (d^-, d^+), (r^-, r^+), P_1, n_1, T_1, p, n, T, L)$ **:**

$v_h \leftarrow$**upper-limit**[a]$(r, (s^-, s^+), (d^-, d^+))$
$((s, d), (P_0, N_0), T_0) \leftarrow$**head**$(T)$
$v_2 \leftarrow$**parameter**$(r, (s, d))$
`WHILE` $P_0 = \{\}$ `DO`: % process non-positive
    $n_1 \leftarrow n_1 + |N_0|$
    $T_1 \leftarrow T_1 \bigcup \{$**head**$(T)\}$
    `IF` $v_2 \leq v_h$: % upper limit not reached
        $l_1 \leftarrow$**make-local**$(a_i, r, (v_1, v_h), (r^-, r^+), P_1, n_1, p, n, (s^-, s^+), (d^-, d^+))$
        $L \leftarrow$**insert-local**(l_1, L)
    `ENDIF`
    $T \leftarrow T \backslash \{$**head**$(T)\}$
    `IF` $T = \{\}$: `RETURN` L
    `ELSE`: $((s, d), (P_0, N_0), T_0) \leftarrow$**head**$(T)$
    $v_2 \leftarrow$**parameter**$(r, (s, d))$
`ENDWHILE`
`WHILE` $N_0 = \{\}$ `DO`: % process non-negative
    $P_1 \leftarrow P_1 \bigcup P_0$
    $T_1 \leftarrow T_1 \bigcup \{$**head**$(T)\}$
    $l_1 \leftarrow$**make-local**$(a_i, r, (v_1, v_2), (r^-, r^+), P_1, n_1, p, n, (s^-, s^+), (d^-, d^+))$
    $L \leftarrow$**insert-local**(l_1, L)
    $T \leftarrow T \backslash \{$**head**$(T)\}$
    `IF` $T = \{\}$: `RETURN` L
    `ELSE`: $((s, d), (P_0, N_0), T_0) \leftarrow$**head**$(T)$
    $v_2 \leftarrow$**parameter**$(r, (s, d))$
`ENDWHILE`
`REPEAT`: % nodes with the same parameter value
    $P_1 \leftarrow P_1 \bigcup P_0$
    $n_1 \leftarrow n_1 + |N_0|$
    $T_1 \leftarrow T_1 \bigcup \{$**head**$(T)\}$
    $l_1 \leftarrow$**make-local**$(a_i, r, (v_1, v_2), (r^-, r^+), P_1, n_1, p, n, (s^-, s^+), (d^-, d^+))$
    $L \leftarrow$**insert-local**(l_1, L)
    $T \leftarrow T \backslash \{$**head**$(T)\}$
    `IF` $T = \{\}$: `RETURN` L
    `ELSE`: $((s, d), (P_0, N_0), T_0) \leftarrow$**head**$(T)$
    $v_3 \leftarrow$**parameter**$(r, (s, d))$
`UNTIL` $v_3 \neq v_2$

`RETURN` $L =$ **all-high-values**$(a_1, r, v_l, (s^-, s^+), (d^-, d^+), (r^-, r^+), P_1, n_1, T_1, p, n, T, L)$

[a] Get the upper limit of the parameter range corresponding to r.

Algorithm D.8: Algorithm for constructing all ranges from a list of REST nodes, given their low limit.

Input: the new local solution: l; the already generated ones: L
Output: the updated set of local solutions: L
insert-local(l, L) **:**
$(a_l, (s_l, d_l), T_l, (P_l, N_l), (prox_l, pur_l)) \leftarrow l$
$f_l \leftarrow$ **fitness**$(prox_l, pur_l)$
$L_1 \leftarrow \{\}$
$h \leftarrow \mathbb{F}$
$\forall l_i \in L$:
    $(a_l, (s_i, d_i), T_i, (P_i, N_i), (prox_i, pur_i)) \leftarrow l_i$
    $f_i \leftarrow$ **fitness**$(prox_i, pur_i)$
    `IF` (**subsumes**$((s_i, d_i), (s_l, d_l)) \wedge prox_l \leq prox_i \wedge N_l \geq N_i$):
        `RETURN` L
    `ELSEIF` (**subsumes**$((s_l, d_l), (s_i, d_i)) \wedge prox_i \leq prox_l \wedge N_i \geq N_l$):
        $L_1 \leftarrow L_1 \bigcup \{l\}$
        $h \leftarrow \mathbb{T}$
    `ELSEIF` $(f_i < f_l \wedge h = \mathbb{F})$:
        $L_1 \leftarrow L_1 \bigcup \{l, l_i\}$
        $h \leftarrow \mathbb{T}$
    `ENDIF`
`IF` $L_1 = \{\}$: `RETURN` $L = \{l\}$
`ELSE`: `RETURN` $L = L_1$

Algorithm D.9: Algorithm to insert a new entry in the set of local solutions.

Undefined ranges

An implicit assumption in the proximity metrics presented in chapter 5 is that the parameter ranges, new and old, are defined. This may not be the case, since any of the range limits is allowed to take the value '?', which is interpreted as an unknown value. If this happens, *proximity* cannot be calculated. For this reason an algorithm is used which replaces undefined range limits with known values in the new or the original parameter range. This algorithm uses a number of priority and consistency constraints to find the appropriate values. The algorithm, Alg. D.10, is an iterative one. In other words, it goes through the four limits more than once, in order to replace them all with numeric values. The rules that it follows are the following:

1. If the low limit of the range is defined do not change it.
2. If both limits of the range are undefined, replace the low limit with the low limit of the second range.
3. If the high limit of the range is defined and the low limit of the second range is not, replace the low with the high limit of the first range.
4. If both the high limit of the first range and the low limit of the second are defined, replace the low limit of the first with the minimum of the two.
5. Use the new low limit of the first range, which may still be undefined, and do the same as above for the low limit of the second range.
6. Similar rules apply for the high limits, but instead of the minimum, the maximum between the low limit of the same and the high limit of the other range is used.

In the worst case, where three of the four limits are undefined, this algorithm will need two iterations to make all undefined limits equal to the defined one.

Input: two ranges, possibly containing undefined limits: (r_1^-, r_1^+) and (r_2^-, r_2^+).
Output: the modified ranges, without undefined limits: (r_1^-, r_1^+) and (r_2^-, r_2^+).

define-limits($(r_1^-, r_1^+), (r_2^-, r_2^+)$**) :**
`IF` $(r_1^-, r_1^+) = (?, ?) \wedge (r_2^-, r_2^+) = (?, ?)$: % all undefined
  `RETURN` $(r_1^-, r_1^+) = (1, 1), (r_2^-, r_2^+) = (1, 1)$
`IF` $r_1^- \neq ? \wedge r_1^+ \neq ? \wedge r_1^- \neq ? \wedge r_1^+ \neq ?$: % all defined
  `RETURN` $(r_1^-, r_1^+), (r_2^-, r_2^+)$
`IF` $r_1^- = ? \wedge r_1^+ = ?$: $r_1^- \leftarrow r_2^-$
`ELSEIF` $r_1^- = ? \wedge r_2^- = ?$: $r_1^- \leftarrow r_1^+$
`ELSEIF` $r_1^- = ?$: $r_1^- \leftarrow \min(r_1^+, r_2^-)$
`ENDIF`
`IF` $r_2^- = ? \wedge r_1^- = ?$: $r_2^- \leftarrow r_2^+$
`ELSEIF` $r_2^- = ? \wedge r_2^+ = ?$: $r_2^- \leftarrow r_1^-$
`ELSEIF` $r_2^- = ?$: $r_2^- \leftarrow \min(r_1^-, r_2^+)$
`ENDIF`
`IF` $r_1^+ = ? \wedge r_1^- = ?$: $r_1^+ \leftarrow r_2^+$
`ELSEIF` $r_1^+ = ? \wedge r_2^+ = ?$: $r_1^+ \leftarrow r_1^-$
`ELSEIF` $r_1^+ = ?$: $r_1^+ \leftarrow \max(r_1^-, r_2^+)$
`ENDIF`
`IF` $r_2^+ = ? \wedge r_1^+ = ?$: $r_2^+ \leftarrow r_2^-$
`ELSEIF` $r_2^+ = ? \wedge r_2^- = ?$: $r_2^+ \leftarrow r_1^+$
`ELSEIF` $r_2^+ = ?$: $r_2^+ \leftarrow \max(r_1^+, r_2^-)$
`ENDIF`
$(r_1^-, r_1^+), (r_2^-, r_2^+) \leftarrow$ **define-limits**$((r_1^-, r_1^+), (r_2^-, r_2^+))$
`RETURN` $(r_1^-, r_1^+), (r_2^-, r_2^+)$

Algorithm D.10: Algorithm to define all limits in two ranges.

D.3 Best-first global search algorithm

The global search algorithm **LRA** combines local solutions into partial and ultimately complete ones, which have the following syntax:

```
<GSol> ::= <LSols>,<Parameters>,<Fitness>
<LSols> ::= <LSol>,<LSols>
<LSols> ::= <LSol>
<Parameters> ::= s | d | nil
```

where `<Fitness>` and `<LSol>` are as before and `<Parameters>` is `nil` if both dimensions of the lowest-level local solution have been examined and `s` or `d` otherwise, i.e., the dimension which has not been examined yet.

Algorithm D.11 describes the high-level process taking place in **LRA**. First the REST is flattened, **flatten-REST**, so that each node holds just a single `(distance,duration)` pair. If the examined event is a repeating one, the algorithm starts by collecting all positive, **get-positive**, and negative, **get-negative**, repeating sequences from the REST. These are represented as pairs of ranges, i.e., rectangular areas in the distance/duration space. It then removes the negative examples that are subsumed, **remove-neg-rep**, and builds the dummy, single-level REST, **dummy-REST**, which just combines the positive and negative examples. If the event is not a repeating one, the pre-processing stage involves only the removal of negative examples, **remove-negative**, which are not of use in the search, i.e. do not interfere with the positive ones and are not covered by the given parameter ranges. Following the pre-processing stage, the positive and negative examples in the REST are counted, **all-examples**, and the **best-first** algorithm is called to perform the best-first search.

Algorithm D.12 describes the **best-first** function, which selects and recursively expands the best partial solution, until a max-fitness, complete solution is constructed. It first calls **LRAloc** to generate the local solutions at the current level of the REST and then adds these local solutions to the partial solutions generated so far, **extend-partial**. If the best solution, L_1, is complete it returns that. If it is not, it selects the local solution at the lowest level of the REST and if this local solution does not provide an answer on both dimensions, **best-first** calls itself recursively with this solution. If the local solution determines both parameter ranges, the nodes that it covers are selected, **nodes-in-range**, and their children are grouped as siblings, **generate-children**. **best-first** is then called recursively to process the children.

Input: the supported event: e_s.

Output: the best complete solution: Sol.

Globals: the set of TCN links: S; the set of RESTs: $\mathcal{T}$.

LRA(e_s) **:**

$(e_s, T) \in \mathcal{T}$
$T_1 \leftarrow$ **flatten-REST**(T)
IF $(a_i, e_i, e_s, ((a_i, s_i), d_i)) \in S$: % repeating event
    $P \leftarrow$ **get-positive**(T_1)
    $N \leftarrow$ **get-negative**(T_1)
    $N \leftarrow$ **remove-neg-rep**(P, N)
    $T_1 \leftarrow$ **dummy-REST**(P, N)
ELSE: % non-repeating event
    $(a_i, e_i, e_s, ((?, (?, ?)), d_i)) \in S$ % terminal subevent
    $T_1 \leftarrow$ **remove-negative**(T_1)
ENDIF
$(p, n) \leftarrow$ **all-examples**(T_1)
$Sol \leftarrow$ **best-first**$(a_i, e_s, \{\mathrm{d}\}, (?, ?), p, n, T_1, \{\}, \{\})$
RETURN Sol

Algorithm D.11: The global refinement algorithm.

Input: examined subevent: a_i; the supported event: e_s; set of non-examined parameters: R; selected range on the opposite dimension: (r^-, r^+); number of positive: p and negative: n examples covered so far; REST: T; the best local solution: Sol; the list of all generated solutions: $Sols$.

Output: the best complete solution: Sol.

Globals: the set of TCN links: S.

best-first$(a_i, e_s, R, (r^-, r^+), p, n, T, Sol, Sols)$ **:**

$L \leftarrow$**LRAloc**$(a_i, R, T, (r^-, r^+), p, n)$
$Sols_1 \leftarrow$**extend-partial**$(Sol,$**tail**$(R), L)$
$Sols \leftarrow Sols_1 \bigcup Sols$
$Sols \leftarrow$**sort-by-fitness**[a]$(Sols)$
$(L_1, R_1, f_1) \leftarrow$**head**$(Sols)$ % the best partial solution
$(a_j, (s_j, d_j), (p_1, n_1), (prox_1, pur_1)) \leftarrow$**head**$(L_1)$[b] % the lowest-level local solution
IF $(R_1 = \{\} \wedge \nexists (a_k, e_k, e_s, ((a_j, s_k), d_k)) \in S)$: $Sol \leftarrow$**head**$(Sols)$ % leaf and complete solution
ELSEIF $R_1 = \{\}$: % non-leaf, both ranges defined
    $(a_k, e_k, e_s, ((a_j, s_k), d_k)) \in S$
    $T_1 \leftarrow$ **nodes-in-range**$(T, (s_j, d_j))$
    $T_2 \leftarrow$**generate-children**(T_1)
    $Sol \leftarrow$**best-first**$(a_k, e_s, \{\mathrm{s,d}\}, (?, ?), p, n, T_2,$**head**$(Sols),$**tail**$(Sols))$
ELSE: % non-leaf, one range defined
    $T_1 \leftarrow$ **nodes-in-range**$(T, (s_j, d_j))$
    $r \leftarrow$**head**(R_1)
    $(r^-, r^+) \leftarrow$**opposite-range**(r, s_j, d_j)
    $Sol \leftarrow$**best-first**$(a_j, e_s, R_1, (r^-, r^+), p, n, T_1, ($**tail**$(L_1), T_1, R_1, f_1),$**tail**$(Sols))$
ENDIF
RETURN Sol

[a] Sorts a set of solutions in descending fitness order.
[b] The first local solution is the last added, i.e., the one at the lowest examined level of the REST.

Algorithm D.12: Global best-first search.

Appendix E

Algorithms for partial supervision

Under partial supervision, a two-stage belief propagation algorithm is used for refinement. The RAPS algorithm, Alg. 6.1, updates incrementally the RESTs, as new input event occurrences are received. This mode of operation is called classification and it involves the recognition of all event occurrences which can be recognised and the forward propagation of model-based recognition beliefs for the event occurrences. Algorithm E.1 in section E.1 describes the updating of the REST. The second mode of operation of the algorithm involves the allocation of feedback from the output to the intermediate nodes of the TCN and the refinement of the model parameters. Algorithm E.2 in section E.2 describes this mode of operation of RAPS.

The causal information needed for feedback allocation is stored in the RESTs. Thus, the syntax of the REST is extended in the following ways:

- There is now a single example list in each node and each element of this list holds the following information:
  ```
  <Example> ::= label,weight,
                subevent-example,subevent-duration,
  ```
  i.e., the label of the example, its weight, the supporting subevent example and its duration. The duration is needed because condensed RESTs are used in the classification stage.
- A proximity value is attached to each duration and distance in the REST. This is to reduce the cost of recalculating recognition beliefs for individual REST branches in the feedback allocation stage.

The recognition beliefs for event occurrences are stored in a global structure, indexed by the event, the example-label and the duration of the example:

```
<RB-set> ::= <Event-RB>,<RB-set>
<Event-RB> ::= event,<RB-list>
<RB-list> ::= <Example-RB>,<RB-list>
<Example-RB> ::= duration,example-number,model-RB,data-RB
```

where `model-RB` is the model-based and `data-RB` the data-based recognition belief for the occurrence.

E.1 Classification algorithm

The basic classification algorithm, Alg. E.1, is very similar to **update-REST** for full supervision. It takes as input the terminal subevent occurrence, the supported event occurrence and

the REST for the supported event, which it updates. In general, no classification information is available and therefore the type of the example is not provided. The algorithm starts by discarding, **satisfy-duration**, the begin points for the terminal subevent occurrence, which do not satisfy the duration window. Then the *cf-prox* for each duration value is calculated and attached to each individual begin point, by **add-cf-to-begin**. Using these begin points a new sub-REST is generated, by **new-REST**, and added to the old REST, by **add-REST**. If the examined event is an output one, **output-event**, its data-based recognition belief is updated, **class-feedback**, taking the value $+1$ for positive occurrences and -1 for negative ones.

Input: terminal event occurrence: $e_i(B, f)$; supported event occurrence: $e_s(b_1, f)$; the REST: T_s.
Output: the updated REST: T_s.
Globals: duration windows: W_d; T-FFCN links: S; recognition beliefs: RB.
RAPSclass$(e_i(B, f), e_s(b_1, f), T_s)$ **:**
$(B_1^+, B_1^-) \leftarrow (\{\}, \{\})$
$(a_i, e_i, e_s, ((?, (?, ?)), d_i)) \in S \vee (a_i, e_i, e_s, ((a_i, s_i), d_i) \in S$ % terminal subevent or repeating
$(a_i, w_{di}) \in W_d$
$B \leftarrow$ **satisfy-duration**$(w_{di}, \{(B, f)\})$
$B \leftarrow$ **add-cf-to-begin**$(\{f\}, d_i, B, f)$
$T_s^* \leftarrow T_s$
`IF` $B \neq \{\}$:
    $T_s' \leftarrow$ **new-REST**[a]$(?, 0, a_i, (B, f), B, e_s(b_1, f))$
    $T_s^* \leftarrow$ **add-REST**(T_s^*, T_s')
`ENDIF`
`IF` $T_s^* \neq T_s$: $xm \leftarrow xm + 1, T_s \leftarrow T_s^*$
`IF` **output-event**(e_s): $RB \leftarrow$ **class-feedback**$(e_s(b1, f), RB)$
`RETURN` T_s

[a]The arguments of **new-REST** are: distance value for the new node, the current depth of the REST, the examined subevent, the stamp for the subevent occurrence, the list of minimum begin points and the supported event occurrence.

Algorithm E.1: The classification stage of the RAPS algorithm.

Under full supervision, **insert-node** is responsible for updating the REST nodes. This function is divided now into two separate procedures: **new-REST** and **add-REST**. The latter procedure simply joins two RESTs with the same format. **new-REST**, calls **new-children**, which returns the children of the new node, with the corresponding sub-RESTs. The example list of the new node contains a single example. Algorithm **new-children** is the equivalent of **add-children** for partial supervision. Their only difference is that there is now no *ctype* information, i.e., no classification for the examples in the example lists. **new-children** calls **add-child** for each occurrence of the preceding subevent which satisfies the duration and distance windows. If there is no preceding subevent, **add-leaf** is called instead. Repeating events are treated as under full supervision.

The only interesting difference of these algorithms with those used under full supervision is in **add-leaf**, which updates the model-based recognition beliefs in the global data structure. During the recursive construction of the new REST, the model-based recognition beliefs for each branch of the REST are attached to the collected minimum begin points. **add-leaf** uses these to create a new entry for the event example in `RB-set`, containing its model-based recognition belief, `model-RB`. The data-based belief, `data-RB`, is initialised to 0.

E.2 Feedback allocation

In the feedback allocation stage, the **RAPSalloc** algorithm processes all of the events in the TCN, in inverse partial support order. In other words, it starts with the output events, performs the intra-REST feedback allocation and refinement of their parameters and allocates feedback to all of the supporting subevent occurrences. Then it moves backwards through the TCN, in a way which ensures that when an event is examined all of the events supported by it have already been processed and allocated feedback to it. For each event being processed, **RAPSalloc** pre-processes the REST, performing the intra-REST feedback allocation, i.e., calculating the example weights. Once this is done, the refinement algorithm, **RAPSref** is called, which is identical to the one used under full supervision. This section describes the intra-REST feedback allocation algorithms.

The intra-REST feedback allocation process in **RAPSalloc**, Alg. E.2, starts by collecting all the "useful examples", **prune-examples**, of the event, i.e., all examples, which have non-zero data-based recognition belief. These examples are stored in a temporary list with a similar format to the `RB-set`, i.e., containing the recognition belief information. Then the REST is flattened, **flatten-REST**, so that each REST node contains a single distance and duration value. At the same time, the model-based recognition beliefs for different REST branches are recalculated. These recognition beliefs, together with the minimum and maximum `model-RB` for each example are used in the rescaling of the example weights. The example list is updated to include the minimum model-based recognition belief for each example, **set-minRB**, and then used in **prune-REST** to prune the REST, so that it contains only the useful examples. **prune-REST** performs also the calculation of the example weights, using the recognition beliefs stored in the example list. Once the weights have been calculated, the refinement algorithm, **RAPSref**, is called to search for the new parameter settings, which are then used to prune the REST even further, **postprune-REST**. The examples, which are excluded by the new parameter values are ignored, in order to reduce the amount of feedback to the supporting subevent occurrence. The inter-REST feedback allocation is done by **REST-feedback**, which simply aggregates the feedback for each supporting occurrence.

Algorithm E.4 describes the process of flattening the REST and calculating the model-based beliefs. It processes each node of the condensed REST and each duration value in each node separately, generating flat nodes. For each node, it starts by flattening the distance list for each child, **flatten-children**, removing also distances that do not satisfy the distance window. **flatten-children** returns the children, T'_c, that correspond to the current duration value and the rest of them, T_c. Then the example list for the flat node is created, **flat-examples**, by looking at the condensed example lists for the current duration value. Each example in the flat example list will also contain the model-based recognition belief of the branch, which is given by the minimum *cf-prox* value in the path so far, CF_b^d. Once the example lists have been created, **flatten-REST** is called recursively to process the children of the new node. **update-**RB_{min} updates the minimum model-based recognition belief for each example.

The flat REST gets pruned by **prune-REST**, Alg. E.3. This algorithm traverses the REST, calculating the duration of each branch. When a leaf node has been reached, it removes, **prune-examples**, from the example list in this node the examples that are not useful to the refinement process, i.e., those which have not received feedback. On the way back to the root node, **parent-examples** selects the examples in a non-leaf node that have been found useful in its children. The example weights are scaled in **prune-examples**, where the feedback value for each example is scaled by the model-based recognition belief of each REST branch.

Input: the examined event: e_i; its REST: T_i; list of events in inverse partial support order: E.

Output: the new parameter settings: Sol.

Globals: distance windows: W_s; T-FFCN links: S; recognition belief lists: $\mathcal{RB}$; the RESTs: $\mathcal{T}$.

RAPSalloc(e_i, T_i, E) **:**

$(a_i, e_i, e_s, ((a_j, s_i), d_i)) \in S \vee (a_i, e_i, e_s, ((a_i, s_i), d_i) \in S$
$(a_i, w_{si}) \in W_s, (e_i, RB_i) \in \mathcal{RB}$
$X_i \leftarrow$**prune-examples**(RB_i)
$(T_i, RB_{min}) \leftarrow$**flatten-REST**$(a_i, 0, w_{si}, T_i, 0, \{\})$
$X_i \leftarrow$ **set-minRB**(X_i, RB_{min})
$T_i \leftarrow$**prune-REST**$(a_i, 0, T_i, X_i, 0)$
$Sol \leftarrow$**RAPSref**(a_i, T_i)
$T_i \leftarrow$**postprune-REST**(Sol, T_i)
$F_i \leftarrow$**allocate-feedback**(T_i)
$\mathcal{RB} \leftarrow$**update-RB**$(F_i, \mathcal{RB})$
`IF` $E \neq \{\}$:
    $e_j \leftarrow$**head**(E), $(e_j, T_j) \in \mathcal{T}$
    $Sol_1 \leftarrow$**RAPSalloc**$(e_j, T_j,$**tail**$(E))$
    `RETURN` $Sol \leftarrow Sol \bigcup Sol_1$
`ELSE: RETURN` Sol

Algorithm E.2: Algorithm for propagating the training feedback to intermediate events.

Input: the currently examined level of the REST: a_i; the current depth in the REST: l; the REST: T_i; the useful examples: X_i; the duration of the examined branch: d_b.

Output: the pruned REST: T_i.

Globals: T-FFCN links: S.

prune-REST(a_i, l, T_i, X_i, d_b) **:**

$T_i' \leftarrow \{\}$
$\forall (s, d, Pos, Neg, T_c) \in T_i$:
    $d_b' \leftarrow d_b + s + d$
    `IF` $T_c = \{\}$: % leaf node
        $(Pos', Neg') \leftarrow$**prune-examples**(Pos, Neg, X_i, d_b', l)
        $T_c' \leftarrow \{\}$
    `ELSE:`
        $l' \leftarrow l + 1$
        $(a_j, e_j, e_s, ((a_i, s_j), d_j)) \in S$ % both repeating and conjunctive
        $T_c' \leftarrow$**prune-REST**$(a_j, l', T_c, X_i, d_b')$
        $(Pos', Neg') \leftarrow$**parent-examples**(T_c', Pos, Neg)
    `ENDIF`
    `IF` $(Pos' \neq \{\} \wedge Neg' \neq \{\})$:
        $T_i' \leftarrow T_i' \bigcup \{(s, d, Pos', Neg', T_c')\}$
    `ENDIF`

`RETURN` T_i'

Algorithm E.3: Algorithm to keep only the REST nodes which contain useful examples.

Input: the currently examined level of the REST: a_i; the current depth in the REST: l; the distance window: w_{si}; the REST: T_i; the duration of the examined branch: d_b; minimum RBs in the path so far: CF_b.

Output: the flat REST: T_i; the minimum model-based belief for each example: RB_{min}.

Globals: distance windows: W_s; T-FFCN links: S.

flatten-REST$(a_i, l, w_{si}, T_i, d_b, CF_b)$ **:**

$T'_i \leftarrow \{\}, RB_{min} \leftarrow \{\}$
$\forall((s, CF^s), D, Pos, Neg, T_c) \in T_i$:
    $\forall(d, CF^d) \in D$:
        $(T_c, T'_c) \leftarrow$**flatten-children**(T_c, w_{si})
        $CF \leftarrow \min(CF^d, CF^s)$
        $d'_b \leftarrow d_b + s + d$
        $(Pos^d, Neg^d, CF^d_b) \leftarrow$**flatten-examples**$(d, d'_b, CF, CF_b, Pos, Neg)$
        `IF` $\exists(a_i, e_i, e_s, ((a_i, s_i), d_i)) \in S$ % repeating
            $l' \leftarrow l + 1$, **repetitions**$(e_i, (n^-_i, n^+_i))$
            $(T'_c, RB'_{min}) \leftarrow$**flatten-REST**$(a_i, l', w_{si}, T'_c, d'_b, CF^d_b)$
            $T'_i \leftarrow T'_i \bigcup \{(s, d, Pos^d, Neg^d, T'_c)\}$
            `IF` $l' > n^-_i$: **update-**$RB_{min}(RB_{min}, \{RB'_{min} \bigcup CF^d_b\})$
        `ELSEIF` $\exists(a_j, e_j, e_s, ((a_i, s_j), d_j)) \in S$: % non-leaf
            $(a_j, w_{sj}) \in W_s$
            $(T'_c, RB'_{min}) \leftarrow$**flatten-REST**$(a_j, l, w_{sj}, T'_c, d'_b, CF^d_b)$
            $T'_i \leftarrow T'_i \bigcup \{(s, d, Pos^d, Neg^d, T'_c)\}$
            $RB_{min} \leftarrow$**update-**$RB_{min}(RB_{min}, \{RB'_{min} \bigcup CF^d_b\})$
        `ELSE`: % leaf
            $T'_i \leftarrow T'_i \bigcup \{(s, d, Pos^d, Neg^d, \{\})\}$
            $RB_{min} \leftarrow$**update-**$RB_{min}(RB_{min}, CF^d_b)$
        `ENDIF`

`RETURN` $(T_i = T'_i, RB_{min})$

Algorithm E.4: Algorithm to flatten the REST.

Appendix F

Sources for marine mammal recordings.

The following is a list of institutes which have archives of whale recordings:

Whale Conservation Institute (WCI). The director of the institute, Dr. Roger Payne, is the person who initiated the systematic analysis of the humpback whale song. Some of his most influential papers can be found in the list of references for this thesis. WCI has a large archive of recordings of humpback and other whales, including most of the material used in the papers by R. Payne, K. Payne, P. Tyack, L. Guinee and others. In addition to recorded songs on tape, they have spectrogram logs and analysis notes for humpback whale songs. The address of the institute is:
Whale Conservation Institute, 191 Weston Road, Lincoln, MA 01773, USA.

Cornell Laboratory of Ornithology. The lab concentrates mainly on bird song. Its Library of Natural Sounds (LNS) contains more than $100,000$ recordings. However, it also runs the Bioacoustics Research Program (BRP), a large part of which is devoted to the analysis of whale sounds. The director of this program, Dr. Christopher Clark, is also heading a research effort named Acoustic Thermometry of Ocean Climate (ATOC), which includes work on Marine Mammal behaviour. Finally, BRP is also involved in the use of the US Navy's Integrated Undersea Surveillance System (IUSS) for the study of whale sounds [71]. The address of the laboratory is:
Cornell Laboratory of Ornithology, PO Box 11, Ithaca, NY 14851, USA.
There is also some information on the WWW:
`http://www.ornith.cornell.edu, http://atoc.ucsd.edu`

University of Oxford, Department of Zoology. Dr. Jonathan Gordon is interested in the automatic classification of whale species by sound and has started an effort to digitise some of the recordings of the WCI. For more information, contact Dr. Gordon or Justin Matthews at:
Department of Zoology, Oxford University, South Parks Road, Oxford, OX1 3PS, UK.

Woods Hole Oceanographic Institution (WHOI). Dr. Peter Tyack and Dr. William Watkins are interested in whale sounds. They can be contacted at:
Biology Department, Woods Hole Oceanographic Institution, Woods Hole, MA 02543, USA.

Appendix G

Frequency ranges for context-sensitive units

Unit	*Session 1*	*Session 2*	*Session 3*	*Session 4*	*Session 5*	*Session 6*	*Session 7*	*Session 8*	*Session 9*	*Session 10*
	(min , max)	(min , max)	(min , max)	(min, max)	(min, max)	(min, max)	(min , max)	(min , max)	(min , max)	(min, max)
a	(250 , 300)	(250 , 300)	(200 , 300)	(200, 300)	(100, 200)	(100, 300)	(100 , 400)	(100 , 250)	(200 , 300)	(200 , 400)
b	(600 , 750)	(550 , 750)	(550 , 750)	(550, 850)	(700, 800)	(700, 800)	(550 , 850)	(550 , 950)	(600 , 700)	(650 , 800)
c	(500 , 600)	(500 , 600)	(450 , 600)	(350, 500)	(450, 700)	(450, 600)	(450 , 600)	(450 , 600)	(350 , 550)	(350 , 500)
d	(600 , 750)	(550 , 650)	(550 , 700)	(550, 750)	(700, 850)	(700, 850)	(650 , 850)	(600 , 700)	(600 , 700)	(600 , 700)
b'	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(550 , 750)	(- , -)	(- , -)	(- , -)
c'	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(450 , 750)	(- , -)	(- , -)	(- , -)
e	(250 , 650)	(300 , 850)	(250 , 900)	(400,1050)	(400,1250)	(350,1050)	(350 , 850)	(400 , 850)	(350 , 850)	(300 , 800)
e'	(- , -)	(- , -)	(- , -)	(450, 750)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
f	(600 , 700)	(550 , 850)	(300 , 750)	(550, 750)	(650, 800)	(350, 850)	(550 , 850)	(450 , 850)	(550 , 750)	(650 , 800)
g	(350 , 500)	(250 , 450)	(250 , 400)	(- , -)	(350, 600)	(250, 450)	(300 , 550)	(300 , 400)	(250 , 400)	(250 , 400)
h	(650 , 850)	(500 , 750)	(600 , 800)	(- , -)	(- , -)	(650, 850)	(850 , 950)	(- , -)	(600 , 750)	(700 , 800)
i	(100 , 300)	(100 , 450)	(100 , 400)	(- , -)	(100, 700)	(100, 700)	(200 , 750)	(100 , 700)	(100 , 700)	(100 , 700)
j	(350 , 450)	(250 , 450)	(200 , 450)	(200, 400)	(250, 550)	(250, 400)	(200 , 550)	(300 , 400)	(250 , 400)	(250 , 500)
k	(650 , 750)	(550 , 750)	(550 , 750)	(750, 850)	(650, 900)	(550, 900)	(450 , 850)	(750 , 900)	(700 , 800)	(800 , 950)
l	(350 , 550)	(300 , 650)	(300 , 750)	(300, 700)	(300, 900)	(250, 800)	(250 , 850)	(350 , 800)	(250 , 650)	(250 , 850)
m	(50 , 100)	(50 , 150)	(50 , 200)	(50 , 150)	(50 , 200)	(50 , 150)	(50 , 200)	(100 , 250)	(100 , 250)	(50 , 150)
n	(100 , 700)	(250 , 550)	(100 , 550)	(100, 700)	(100, 650)	(100, 700)	(250 , 750)	(100 , 550)	(250 , 550)	(100 , 600)
o	(350 , 550)	(350 , 650)	(300 , 550)	(300, 550)	(300, 450)	(350, 550)	(350 , 600)	(350 , 550)	(250 , 400)	(250 , 400)
p	(300 , 400)	(250 , 350)	(300 , 550)	(250, 500)	(250, 400)	(250, 450)	(300 , 550)	(350 , 500)	(250 , 400)	(250 , 450)
q	(100 , 700)	(250 , 500)	(100 , 550)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
r	(350 , 550)	(350 , 550)	(250 , 550)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
s	(300 , 400)	(300 , 400)	(400 , 600)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
t	(450 , 550)	(450 , 550)	(850 ,1550)	(450, 550)	(400, 550)	(450, 600)	(550 , 600)	(650 , 750)	(700 , 800)	(300 , 450)
t'	(1050,2050)	(1550,2350)	(50 , 250)	(- , -)	(850,1450)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
t''	(- , -)	(550 ,2350)	(1550,2150)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)	(- , -)
u	(100 , 200)	(100 , 400)	(50 , 250)	(200, 300)	(100, 250)	(100, 250)	(350 , 600)	(350 , 550)	(250 , 400)	(250 , 450)
v	(100 , 200)	(100 , 400)	(50 , 250)	(200, 300)	(100, 250)	(100, 250)	(350 , 600)	(350 , 550)	(250 , 400)	(250 , 450)
w	(450 , 550)	(450 , 550)	(450 , 600)	(450, 550)	(550, 650)	(450, 550)	(450 , 550)	(650 , 750)	(750 , 950)	(400 , 550)
x	(800 ,1550)	(800 ,2050)	(750 ,1550)	(700,1550)	(750,1550)	(750,1650)	(1050,1550)	(1550,2250)	(1550,2050)	(850 ,1550)
y	(250 , 350)	(100 , 300)	(50 , 250)	(200, 300)	(50 , 250)	(100, 250)	(350 , 600)	(350 , 550)	(250 , 400)	(250 , 450)
z	(250 , 350)	(100 , 300)	(50 , 250)	(200, 300)	(50 , 250)	(100, 250)	(350 , 600)	(350 , 550)	(250 , 400)	(250 , 450)