Focused Crawling using Temporal Difference-Learning

Alexandros Grigoriadis[1,2], Georgios Paliouras[1]

[1] Software and Knowledge Engineering Laboratory
Institute of Informatics and Telecommunications,
National Centre for Scientific Research "Demokritos"
153 10 Ag. Paraskevi, Athens, Greece
{grigori, paliourg}@iit.demokritos.gr

[2] Language Technology Group
Human Communication Research Centre
University of Edinburgh, Edinburgh, UK

Abstract. This paper deals with the problem of constructing an intelligent Focused Crawler, i.e. a system that is able to retrieve documents of a specific topic from the Web. The crawler must contain a component which assigns visiting priorities to the links, by estimating the probability of leading to a relevant page in the future. Reinforcement Learning was chosen as a method that fits this task nicely, as it provides a method for rewarding intermediate states to the goal. Initial results show that a crawler trained with Reinforcement Learning is able to retrieve relevant documents after a small number of steps.

Keywords: Machine learning, reinforcement learning, web mining, focused crawling.

1 Introduction

World Wide Web can be considered as a huge library of every kind of information, accessible to many people throughout the world. However, it lacks a global indexing system that would consist of an explicit directory of all the information found in the Web. In order to deal with this problem, many Web tools have been constructed that mostly try either to construct a Web directory a priori, or respond to a user's query about keywords contained in a Web page.

These methods usually require exhaustive crawling, an effort to traverse as many Web pages as possible in order to maintain their database updated. However, this procedure is very resource consuming and may take weeks to be completed. On the other hand, "Focused Crawling" [3] is the effort to retrieve documents relevant to a predefined topic, trying to avoid irrelevant areas of the Web. Therefore it is more effective in finding relevant documents faster and more accurately.

A "Focused Crawler" searches the Web for relevant documents, starting with a base set of pages. Each of these pages contains usually many outgoing hyper-

links and a crucial procedure for the crawler is to follow the hyperlinks that are more probable to lead to a relevant page in the future. Therefore, the crawler must include a component that evaluates the hyperlinks, usually by assigning a numerical "score" to each one of them. The highest the score is, the more probable it is that this hyperlink will lead to a relevant page in the future.

This component, the "Link Scorer" is implemented here by a reinforcement learning (R.L.) agent. An R.L. agent can recognize different states of the environment and for each of these *states* $s \in S$ it is able to choose an *action* α from a set of actions A. The choice of the action that the agent will perform in a specific state, is based on the *policy* π of the agent and can be represented simply as a look-up table.

Except for the agent, another important factor of an R.L. scheme, is the environment. The environment "judges" each of the agent's choices (actions) by providing a numerical *reward*. The reward is indicative of what we want the agent to perform, but not how it will perform it.

Based on the rewards it receives, the agent's policy is rearranged towards the optimal policy π^*. When a reward is given, the course of actions that the agent has followed so far gets credit. The way this credit is distributed backwards to the actions is determined by the specific R.L. method adopted. Moreover, the environment makes the transition to the next state s_{t+1}, given the current state s_t and the action a_t, chosen by the agent.

Reinforcement learning seems to fit nicely to the task of focused crawling. Indeed, the environment can tell the agent when it has done a good job (found a relevant page), but not how to do it - this is its own responsibility. Moreover, when the agent receives a reward the whole course of actions followed is affected, and not only the last one as would be the case in a supervised learning approach. This is a promising solution to the central problem of focused crawling, which is to assign credit to all the pages of the path that leads to a relevant document.

Our aim is to construct a focused crawler that uses an R.L. agent to train the "link scoring" component. This crawler should have increased ability to identify good links, because of the R.L. scheme, and therefore become more efficient and faster than a baseline crawler.

The next section presents a survey of the most important related work on Focused Crawling. Special attention is paid to methods engaging machine learning and the different aspects of dealing with this problem are illustrated. Section 3 is devoted to our own approach and the issues of representing the entities of the problem in an R.L. scheme. Section 4 describes our implementation of the R.L. agent and section 5 presents experimental results. These results are analyzed, in order to draw conclusions on our method, which are presented in the last section.

2 Related Work

The first attempts to implement focused crawling were based on searching the Web using heuristic rules that would guide the choices of the crawler. These rules are usually based on keywords found near the link and in the rest of the

page that contains it. The crawler performs a search strategy combined with the heuristic rules in order to follow successful paths leading to relevant pages. Such implementations are "Fish-Search" [6] and "Shark-Search [8].

More recent methods use information related to the structure of the Web graph, in order to perform more efficient focused crawling. Some of these methods take advantage of the "Topical Locality" of the Web (the property of pages with similar topic being connected with hyperlinks [2]) and use it to guide the focused crawler [3]. Moreover, the "backlink" information (pages that link to a certain document), provided by search engines like Google or Altavista, can be used to generate a model of the Web-graph near a relevant page, such as in the case of "Context Graphs" [7]. Finally, information such as contents of in-linking pages, tokens in the URL, and contents of sibling pages, can be extracted in order to train an agent to recognize the "linkage structure" for each topic [1].

There are also some methods that use R.L. in order to deal with focused crawling. In [9], the crawling component is based on R.L., although some simplifying assumptions are made. More specifically, in this approach the state space has been omitted, due to high dimensionality of the data. Therefore, the agent examines only the value of the possible actions to be taken, irrespective of the state of the environment. The actions are represented by the different hyperlinks that exist in a Web page, and the value of each action is estimated by a "bag-of-words" mapping of the keywords in the neighborhood of the hyperlink to a scalar value.

3 Problem Representation

In order to analyze the issues that arise in the representation of the focused crawling task as an R.L. task, we should examine a small part of the Web graph, like the one depicted in **Figure 1**. Each node ①..⑨ represents a Web page and each arc represents a link from a Web page to another. Web page ⑨ is relevant and there is only one path, following the nodes ①→④→⑧→⑨ leading to that page.

The aim of Focused Crawling, is to be able to recognize promising links early on, in order to follow the right path. Assume that an agent is in node ④ and has to choose between two links to follow, link 1 and link 3. It should be able to evaluate those links and choose the best, which is the one that is more promising in leading to a relevant page. In this case it should be link 1. By following this link, the agent will now be in node ⑧, which is one step closer to the relevant page.

Reinforcement learning seems to fit this task nicely. When the agent finds the target, which in this case is the relevant page, all the actions that lead to this take credit, allowing the agent to learn patterns of paths leading to relevant pages in the Web. However, a great deal of attention must be paid to the design of the reinforcement learning approach, in order to determine the most suitable problem representation, the role of each unit and the environment's behaviour.

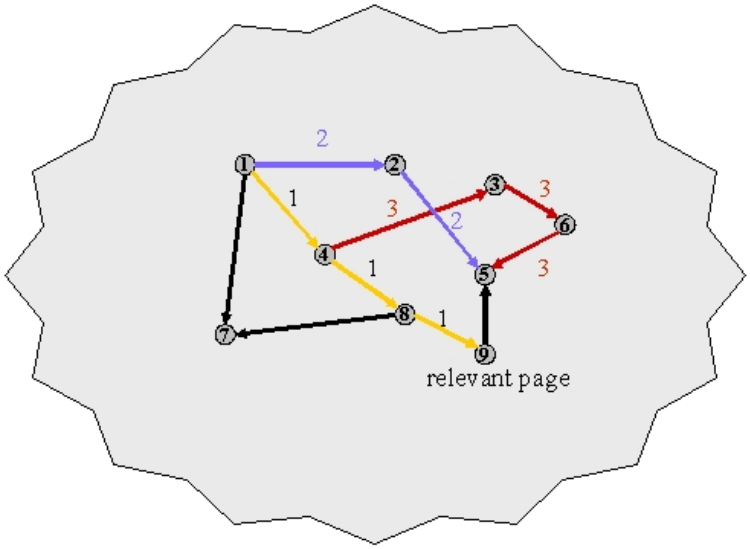

Fig. 1. A small part of the Web graph

In our approach, every Web page represents a different state $s_t \in S$. The set of actions contains the hyperlinks that exist in each page. Therefore, the agent being in state s_t (Web page), must choose among the actions that exist for this state $\alpha \in A(s_t)$, i.e. the hyperlinks found in this Web page. This action leads to another state s_{t+1} and a numerical reward, r_{t+1}, is given to the agent. This reward is +1 in case the Web page the agent has moved to is relevant, and 0 otherwise.

The aim of the R.L. agent is to maximize the reward it accumulates over the long run. This quantity is called the *Return*, and is defined as follows:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1}$$

where γ is a discount factor, denoting the importance of recent rewards compared to older ones.

In order to find a policy, i.e. a mapping from states to actions, that would maximize the *Return*, the agent must be able to evaluate each state according to that criterion, as follows:

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\} \tag{2}$$

which is called the state-value function for policy π. In our case, the state-value function represents the possibility of a Web page being on a path to a relevant page. Therefore, a page with high state-value is preferable to a page with a lower

one. When the agent must make a decision upon which hyperlink to be followed, it needs to estimate the state-value of the page pointed to by the hyperlink, termed the *outlink* page here. In other words, being in state s_t the agent needs to find the action a_t that leads to the state s_{t+1} with maximum value:

$$\arg\max_{a_t} V(s_{t+1}) \tag{3}$$

This estimation can be achieved either by estimating the values of all the possible next states, e.g. by fetching and evaluating all the outlink pages, or by estimating the value of the actions themselves, i.e. evaluating the hyperlinks, rather than the pages they point to.

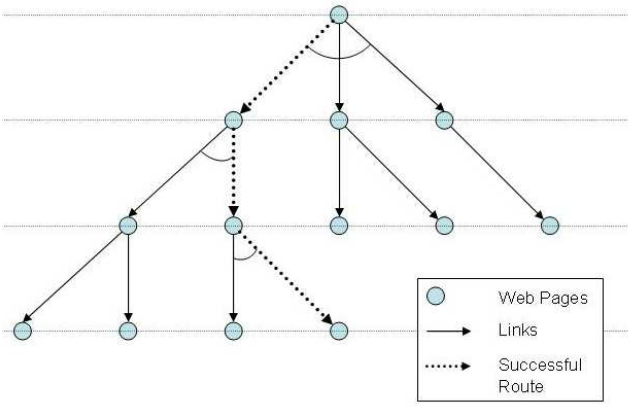

Fig. 2. A search tree of Web pages

This process is illustrated schematically in **Figure 2**, which depicts the outlink structure by a tree and our purpose is to find the best search strategy for relevant pages. The successful route is denoted by a dotted line. Starting from the root page (level 0) the agent needs to evaluate the pages on the next level and choose the best, according to the value function. Being in a node at level n it evaluates only the children of this node at level $n + 1$. As the experience of the agent grows it will become easier to find the right path in an efficient and cost-effective way.

4 Implementation

In order for an R.L. agent to be implemented, there are many practical issues that need to be considered. One is the dimensionality of the state-space. In our case,

each state (Web page) is represented by a feature vector of 500 binary values. Each value corresponds to the existence or not of a specific keyword, which is important for the classification of a page as relevant or not. This makes up a space of approximately $3 \cdot 10^{150}$ different states, that can not be examined separately in a tabular policy format. Therefore, a function approximation method must be employed, where the features of each state are used as the input, and the estimation of the state-value as the output of the function.

The method chosen for our experiments was Temporal Difference Learning with eligibility traces TD(λ) and gradient descent function approximation [11]. Temporal Difference is a very commonly used method for R.L. Eligibility traces are used to implement TD(λ), a faster version of TD, where a fewer number of episodes is required to train the agent.

Moreover, a neural network is trained to estimate the values of different states, since their dimensionality does not allow a direct mapping. This neural network receives a training instance at each step of the crawling process. The input vector represents the features of the current Web page and the output the estimated state-value of that page, based on the reward that is received. The reward takes the value 1 or 0 according to whether the page is relevant or not. Implementing TD(λ), each synapse is associated with its weight and its *eligibility trace*, which captures the discounted reward provided by the R.L. policy. These parameters, weights and eligibility traces, are updated, in order to ensure that the network gives credit to all the actions of a successful course.

The agent operates in two modes: "training" and "crawling". During "training" the agent executes a number of episodes, usually from 1000 to 10000, starting from a root page and following hyperlinks randomly, until it completes a number of steps (e.g. 10), or until it finds a relevant page. At each step, the agent is in state s_t and performs action α_t, receiving a reward r_{t+1} according to how good the action was (led to a relevant page or not). The reward r_{t+1} along with the features representing state s_t are fed to the neural network. Since the neural network is enhanced with eligibility traces, it gradually learns to evaluate a state's potential of leading to a relevant page, not only immediately but also in the future.

In the "crawling" mode, the agent is embedded in a crawler, which maintains a list of hyperlinks and their scores. Starting from a "root" page, the crawler evaluates all the outlinks using the trained neural network. These hyperlinks with their scores are added to the list. The crawler selects the hyperlink with the highest score, examines whether it is relevant or not, and extracts and evaluates their outlinks in order to store them in the list. The process ends when a predefined number of pages have been visited.

In order to evaluate outgoing links at each step, there are two alternative approaches that can be followed. The first is to fetch all the outlink pages and estimate their state-values. This is referred to hereafter as the "original lookahead method". However, this one-step lookahead causes a computational overhead, because the crawler is obliged to fetch all the outlink pages, even though it may decide not to follow most of them. Since the performance of a crawler is usually

measured according to the number of pages that have to be visited until the relevant pages are found, this overhead cannot be ignored. Therefore, it would be desirable to have a variant of the original method, that is able to assign scores to links without having to visit them first.

This is realised by using the score of the current page as an approximation of the score of the pages that it links to. The crawler first examines the root page and assigns a score to it using the same procedure as in the original method. However, it does not fetch outlink pages and examine their contents. Instead, they immediately inherit their parent's score before being added to the list. Then, the crawler chooses the page with the highest score and visits it. When a page is found that has already been scored and needs to inherit a new value (multiple inheritance from more than one parents), the average of all the previous scores is used. This approach is referred to hereafter as the "variant without lookahead".

5 Experiments

5.1 Setup

The data used for the experiments are those used in the 2^{nd} domain of the multilingual information integration project CROSSMARC [4, 10], for the English and Greek language. CROSSMARC examined two thematic domains: "laptop product descriptions" and "job adverts on corporate Web sites". The latter domain is considered here.

The datasets used in the experiments represent Web sites containing pages of the specific domain for the two languages. The characteristics of the datasets are shown in **Table 1** and **Table 2**.

One characteristic of the domain that makes it particularly challenging for a focused crawler is the small proportion of relevant pages in each dataset. This situation, however, is very realistic, given the vastness of the Web, in which a crawler operates. Furthermore, it should be noted that the Greek datasets are generally larger and contain more relevant pages.

In order to present objective and comparative results, cross-validation is used according to the following procedure:

- Given n different datasets, a separate Neural Network with eligibility traces is trained on each one of them.
- After the training has been performed, each dataset passes through the crawling phase as follows:
  - The selected dataset is crawled, using an average of the value functions of the $n-1$ remaining Neural Networks.
  - The crawler's performance is calculated as a cumulative count of the number of relevant pages found at each navigation step.
- The procedure continues until all the datasets have been tested.

Table 1. Datasets used in experiments - English

#	Name	Web Pages	Hyperlinks	Average Outlinks	Relevant Pages
1	ApcInc	24	342	14	1
2	En-Vivo	24	88	4	1
3	Rowan	10	60	6	1
4	Harmonia	241	3332	14	1
5	Quarry	92	1712	19	9

Table 2. Datasets used in experiments - Greek

#	Name	Web Pages	Hyperlinks	Average Outlinks	Relevant Pages
1	Abc	194	1544	8	11
2	Forthnet	1907	4480	2	18
3	Intracom	555	15599	28	11
4	Marac	87	1249	14	3
5	Sena	102	2074	20	1

5.2 Experimental Results

The experiments were run for 1000 episodes with a maximum of 10 steps each. **Figures 3 to 6** depict the percentage of the relevant pages that were found by the algorithm against the percentage of the pages visited, for the various methods. Each point represents the number of pages that have been examined so far (x-axis) and the number of relevant pages that were discovered until then (y-axis). Therefore, lines positioned in the left side of the graph represent better performance (more relevant pages found earlier). Also, the fewer relevant pages a dataset has, the steeper the line is, since there are less points in the graph that denote the discovery of a relevant page.

Figures 3 and 4 present the results for the English sites. Both methods perform better in the "Quarry" dataset, followed by "En-vivo", "ApcInc", "Harmonia" and finally "Rowan". Although the variant method performed worse in the "Quarry" dataset than the original one, it was better in the other datasets. However, since "Quarry" was the only dataset containing more than one relevant pages, it represents a more realistic situation, while the other datasets can be considered problematic.

Figures 5 and 6 present the graphs for the Greek sites. The original method performed better for the "Forthnet" dataset, while the variant in all the other datasets. Moreover, the Greek datasets are much larger, with various graphi-

cal structures (number of outlinks) and thus pose a more realistic evaluation scenario.

Using the same datasets used in CROSSMARC, which is a variant of the method presented in [9], produced the results shown in **Figures 7 and 8**. In the English datasets, CROSSMARC's crawler performed better on large datasets, such as 'Harmonia", and "Quarry", while its performance was worse in the rest of the datasets, being worst in the "En-Vivo" case. For the Greek datasets, the performance of the CROSSMARC's crawler is similar to our method, being better in some datasets and worse in others.

It should be noted that the results for the original lookahead method are not directly comparable with the other two methods. This is because the lookahead method has to visit more pages en route to the relevant page.

Despite this fact the lookahead method seems to be worse than the other two methods in most cases. Therefore, the additional computation is not justified. Among the other two methods, no clear conclusion can be drawn about which of the two is better. However, the fact that the two methods are based on the R.L. principle, combined with the fact that they seem to complement each other in terms of performance, indicates a potential synergy among them.

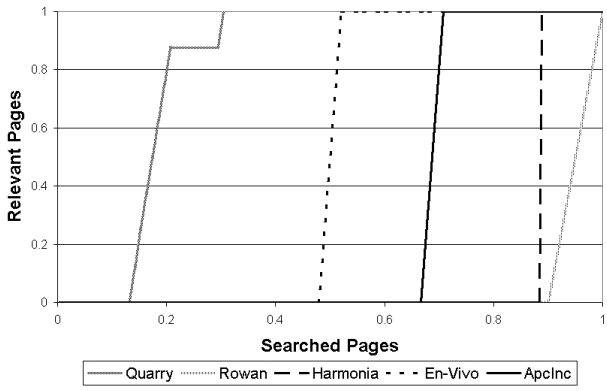

Fig. 3. Results for the original lookahead method - English

6 Conclusion

This paper dealt with the problem of Focused Crawling using an intelligent crawling agent based on Reinforcement Learning. A crawler must be able to recognize patterns within the Web graph and make the right choices in order to be efficient and cost-effective. Reinforcement learning was chosen because it is a method that allows an agent to accumulate knowledge by experimenting with the environment, without using direct supervision. It seems to be appropriate

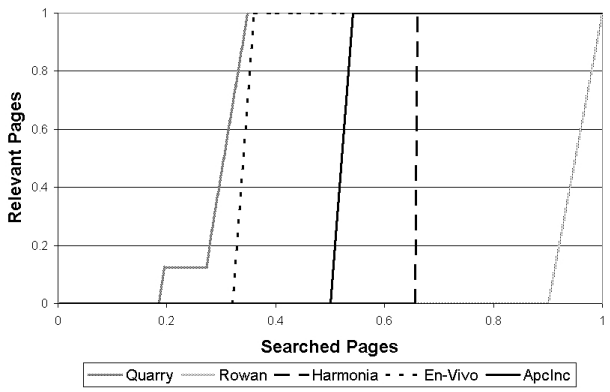

Fig. 4. Results for the variant without lookahead- English

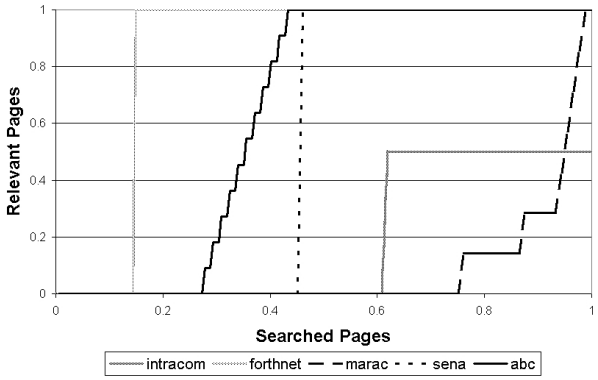

Fig. 5. Results for the original lookahead method - Greek

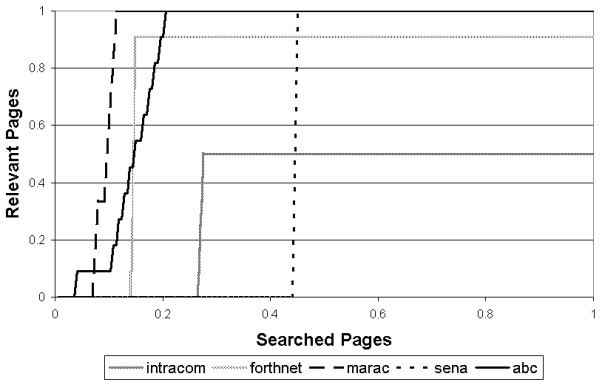

Fig. 6. Results for the variant without lookahead - Greek

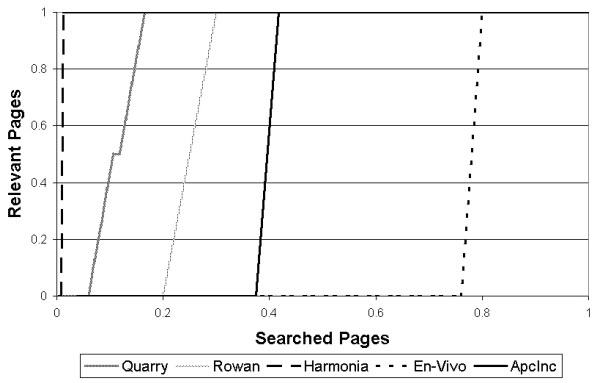

Fig. 7. CROSSMARC's Crawler - English

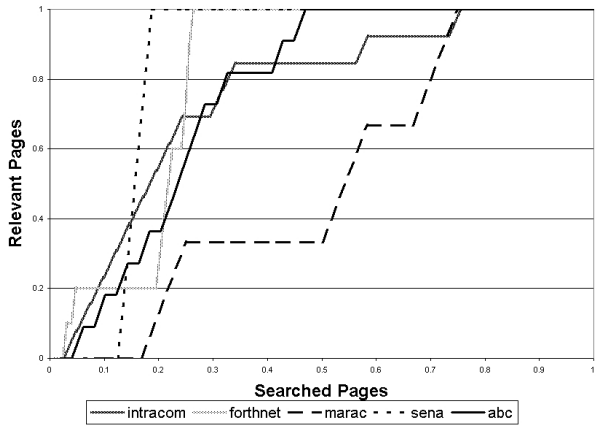

Fig. 8. CROSSMARC's Crawler - Greek

for the task of Focused Crawling where success can be recognised but detailed guidance to this success cannot be provided, as would be required by a supervised learning approach.

The results of the experiments show that reinforcement learning is a good choice for this task. Indeed, in most of the cases only a small number of steps was required in order to retrieve all the relevant pages.

Further work includes further experimentatin and potential extension of the method, incorporating features of the method used in CROSSMARC's crawler.

References

1. Aggarwal C., Al-Garawi F. and Yu P. *Intelligent Crawling on the World Wide Web with Arbitrary Predicates.* In Proceedings of the 10th International WWW Conference, pp. 96-105, Hong Kong, May 2001.
2. Brin S. and Page L. *The Anatomy of a Large-Scale Hypertextual Web Search Engine.* In the Proceedings of the Seventh International WWW Conference, pp. 107-117, Brisbane, April 1998.
3. Chakrabarti S., van den Berg M. and Dom B. *Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery.* In Proceedings of the 8th International WWW Conference, pp. 545-562, Toronto, Canada, May 1999.
4. CROSS-lingual Multi Agent Retail Comparison. `http://www.iit.demokritos.gr/skel/crossmarc`.
5. Karkaletsis V., Paliouras G., Stamatakis K., Pazienza M.-T., Stellato A., Vindigni M., Grover C., Horlock J., Curran J., Dingare S. *Report on the techniques used for the collection of product descriptions*, CROSSMARC Project Deliverable D1.3, 2003.
6. De Bra P., Houben G., Kornatzky Y. and Post R. *Information Retrieval in Distributed Hypertexts.* In Proceedings of the 4th RIAO Conference, pp. 481-491, New York, 1994.
7. Diligenti M., Coetzee F.M., Lawrence S., Giles C.L. and Gori M. *Focused Crawling Using Context Graphs.*VLDB 2000, Cairo, Egypt, pp. 527-534, 2000.
8. Hersovici M., Jacovi M.,Maarek Y., Pelleg D., Shtalhaim M. and Sigalit U. *The Shark-Search Algorithm - An Application: Tailored Web Site Mapping.* In Proceedings of the Seventh International WWW Conference, Brisbane, Australia, April 1998.
9. McCallum A., Nigam K., Rennie J. and Seymore K. *Building Domain-Specific Search Engines with Machine Learning Techniques.* In AAAI Spring Symposium on Intelligent Agents in Cyberspace, Stanford University, USA, March 1999.
10. Stamatakis K., Karkaletsis V., Paliouras G., Horlock J., Grover C., Curran J.R. and Dingare S. *Domain-specific Web Site Identification: The CROSSMARC Focused Web Crawler.* In Proceedings of the Second International Workshop on Web Document Analysis (WDA2003), p.75-78. 3-6 August 2003, Edinburgh, Scotland.
11. Sutton R., Barto A. *Reinforcement Learning. An Introduction.* MIT Press, Cambridge, MA (2002).