Multiple-goal Search Algorithms and their Application to Web Crawling

Dmitry Davidov and **Shaul Markovitch**
Computer Science Department
Technion, Haifa 32000,Israel
email:dmitry,shaulm@cs.technion.ac.il

Abstract

The work described in this paper presents a new framework for heuristic search where the task is to collect as many goals as possible within the allocated resources. We show the inadequacy of traditional distance heuristics for this type of tasks and present alternative types of heuristics that are more appropriate for multiple-goal search. In particular we introduce the yield heuristic that estimates the cost and the benefit of exploring a subtree below a search node. We present a learning algorithm for inferring the yield based on search experience. We apply our adaptive and non-adaptive multiple-goal search algorithms to the web crawling problem and show their efficiency.

Introduction

WWW search engines build their indices using brute-force crawlers that attempt to scan large portions of the web. Due to the size of the web, these crawlers require several weeks to complete one scan, even when using very high computational power and bandwidth (Brin & Page 1998). Many times, however, there is a need to retrieve only a small portion of the web of pages dealing with a specific topic or satisfying some user criteria. Using brute-force crawlers for such a task would require enormous resources with most being wasted on non-relevant pages. Is it possible to design a smart crawler that would scan only relevant parts of the web and would retrieve the desired pages using much less resources than the exhaustive crawlers?

Since the web can be viewed as a large graph (Kumar *et al.* 2000), where pages are nodes and links are arcs, we may look for a solution to the problem of selective crawling in the field of heuristic graph-search algorithms. A quick analysis, however, reveals that the problem definition assumed by the designers of heuristic search algorithms is different than the setup for selective crawling which makes them inappropriate for the given task. The crucial difference between the two setups is the success criteria. In both setups there is a *set* of goal states. In the heuristic search setup, however, the search is completed as soon as a single goal state is found, while in the selective crawling setup, the search continues to collect as many goal states as possible within the given resources.

One of the reasons why common search techniques are not applicable to multiple-goal search is that most are based on a heuristic function that estimates the distance from a node to the nearest goal node. Such heuristics are not useful for multiple-goal search. To understand why, consider the search graph described in Figure 1.

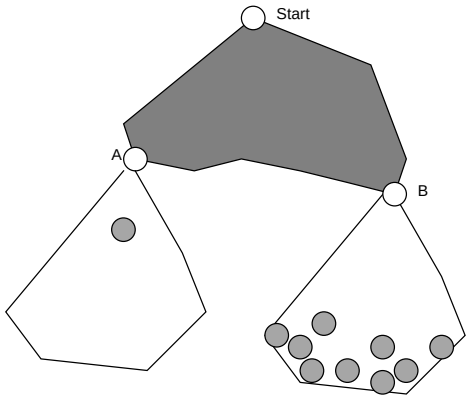

Figure 1: Using distance heuristic for a multiple-goal problem.

The grey area is the expanded graph. Assume that we evaluate nodes A and B using a distance-based heuristic. Obviously node A has a better heuristic value and will therefore be selected for pursuing. This is indeed a right decision for traditional search where the task is to find *one* goal. For multiple-goal search, however, B looks a much more promising direction since it leads to an area with a high density of goal nodes.

In this research we define a new class of graph-search problems called *multiple-goal search* and develop algorithms to deal with such problems. We define a new class of heuristic functions, called *yield* heuristics, and show that they are more appropriate for multiple-goal search than the traditional distance-estimation heuristics. We show search algorithms that use such heuristics and present learning mechanisms for acquiring such heuristic functions from experience. Finally, we show the application of the framework to the selective crawling and present results of an experimental study in the web domain.

The multiple-goal search problem

The multiple-goal search problem is similar to the single-goal search problem in several aspects. We assume that we are given a finite set of start states, an expand function that gives the successors of every state and a goal predicate. We assume that we search for goals by repeatedly expanding states. The main difference between our framework and the traditional one is the stopping criteria. In the traditional single-goal search framework the search algorithm stops as soon as it encounters a goal state (or, in the case of optimizing search, as soon as a lowest-cost path to a goal state is found). In our multiple-goal search framework we define two alternative stopping criteria for the algorithm:

- When it spends a given allocation of resources.
- When it finds a given portion of the total goals.

The performance of multiple-goal search algorithm is evaluated using two alternative methods corresponding to the above stopping criteria. In the first case, we evaluate the performance by the number of goal states found using the given resources. In the second case we evaluate the performance by the resources spent for finding the required portion. An alternative approach would be to treat the search algorithm as *anytime algorithm*. In such a case we evaluate the algorithm by its performance profile.

Heuristics for multiple-goal problems

In the introduction we have shown an example that illustrates the problem of using traditional distance-estimation heuristic for multiple-goal search. One of the main problems with using such a distance heuristic is that it does not take into account goal concentration but only the distance to the nearest goal. This can lead the search into a relatively futile branch such as the left branch in Figure 1 instead of the much more fruitful right branch.

There are several possible alternatives for this approach. If the set of goal states, S_G, is given explicitly, we can use the distance heuristic to estimate the sum of distances to S_G. Such heuristic will prefer the right branch of the search graph in Figure 1 since moving in that direction reduces the sum of distances significantly. If S_G is not given we can try to build an estimator for the sum-of-heuristic distances function. Sometimes, instead of getting S_G, we may be supplied with a set of heuristic functions that estimate the distances to individual members or subsets of S_G. In such a case we can sum the values returned by the given set of heuristics. One problem with the *sum* heuristic is its tendency to try to progress towards all the goals simultaneously. Thus, if there are groups of goals scattered around the search front, all the states around the front will have similar heuristic values. Each step reduces some of the distances and increases some of them leading to a more-or-less constant sum.

One possible remedy to this problem is giving higher weight to progress. As before, we assume that we have either the explicit goal list S_G or, more likely, a set of distance heuristics to goals or goal groups (categories). Instead of measuring the global progress towards the whole goal set, we measure the global progress towards each of the goals or goal groups and prefer steps that lead to progress towards more goals. More formally, we use the heuristic $H(n) = |\{g \in S_G | h(n,g) < \min_{c \in Closed} h(c,g)\}|$ where $Closed$ is the list of already expanded nodes. We call this strategy *front advancement*.

Another approach for multiple-goal heuristic is to deal explicitly with the expected cost and expected benefit of searching from the given node. Obviously, we prefer subgraphs where the cost is low and the benefit is high. In other words, we would like high return for our resource investment. We call a heuristic that tries to estimate this return a *yield* heuristic.

For a finite graph, we can estimate the yield by the following pessimistic estimation $Y(n, S_G) = |N \cap S_G| / |N|$ where N is set of all nodes which can be reached from n. This estimation assumes that the whole subgraph is traversed in order to collect its goals. Alternatively we can estimate the yield by the optimistic estimation $Y(n, S_G) = |N \cap S_G| / |g(n, N, S_G)|$ where N is defined as above and $g(n, N, S_G)$ are the nodes of a directed graph with minimal number of edges such that it contains a directed path from n to each node in $N \cap S_G$. This estimation assumes that we do not waste resources unnecessarily when collecting $N \cap S_G$. The above two definitions can be modified to include a depth limit d. Then, instead of looking at the graph N of all nodes reachable from n we restrict the definition to nodes that are at depth of at most d from n.

Naturally, it is very difficult to build yield estimation heuristics. In the following sections we will show how such heuristics can be adapted.

Heuristic search for multiple-goal problems

Assuming that we have a heuristic function suitable for multiple goal search as described above, we can now use this heuristic in traditional heuristic search algorithms such as best-first search. The main difference, in addition to the different type of heuristic, is that when a goal is encountered, the algorithm collects it and continues (unlike the single-goal best-first search which would stop as soon as a goal is found). Figure 2 shows the multiple-goal version of best-first search.

```
procedure MGBestFirst(StartNodes,RLimit)
    Open ← StartNodes Closed ← {}
    Goals ← {} RCount ← 0
    loop while Open ≠ {} AND RCount < RLimit
        let n be a node with minimal h(n) in Open
        Open ← Open \{n} ; Closed ← Closed ∪{n}
        RCount ← RCount + 1
        For each n' ∈ Succ(n)
            If n' ∉ Closed ∪ Open
                If G(n') then Goals ← Goals ∪ {n'}
                Open ← Open ∪ {n'}
    Return Goals
```

Figure 2: The multiple-goal best-first search algorithm

One possible side-effect of not stopping when discovering

goals is the continuous influence of the already-discovered goals on the search process. The found goals continue to attract the search front while we would have preferred that the search would progress towards undiscovered goals. The problem is intensified when the paths leading to undiscovered goals pass through or nearby discovered goals.

One possible solution to this problem is to eliminate the effect or reduce the weight of discovered goals in the heuristic calculation. Thus, for example, if the set of goals (or a set of distance-heuristic functions to goals) is given, the discovered goals can be removed from the heuristic calculation. If a set of heuristics to subsets of S_G is given, then we can reduce the weight of the subset (which may be defined by a category) of the discovered goal.

Learning yield heuristics

While it is quite possible that yield heuristics will be supplied by the user, in many domains such heuristics are very difficult to design. We can use learning approach to acquire such yield heuristics online during the search. We present here two alternative approaches for inferring yield:

1. Accumulate partial yield information for every node in the search graph. Assume that the yield of the explored part of a subtree is a good predictor for the yield of the unexplored part.
2. Accumulate yield statistics for explored nodes. Create a set of examples of nodes with high yield and nodes with low yield. Apply an induction algorithm to infer a classifier for identifying nodes with high yield.

Inferring yield from partial yield

The *partial* yield of a node n at step t of executing multiple-goal search algorithm is the number of goals found so far in the subtree below n divided by the number of nodes generated so far in this subtree. We use the partial yield of a node to estimate the expected yield of its brothers and their descendants. For each expanded node n we maintain D partial yield values for depth from 1 to D where D is a predefined depth limit. For computing the partial yield we keep in the node, for each depth, the number of nodes generated and the number of goals discovered so far to this depth. When generating a new node we initialize the counters for the node and recursively update its ancestors up to D levels above. The algorithm is described in Figure 3. To avoid updating an ancestor twice due to multiple paths, one must mark already updated ancestors. For simplicity we listed the algorithm without this mechanism. The estimated yield of a node is the average yields of its supported children – those are children with sufficiently large expanded subtrees. If there are no such children, the yield is estimated (recursively) by the average yield of the node parents. The depth values are adjusted appropriately. The algorithm is listed in Figure 4. For simplicity we avoided listing the handling of special cases. When calling the yield estimation algorithm, we can adjust the depth parameter according to the resource limit or to the number of goals required. For example, we can use similar algorithm to estimate the tree sizes and require depth for which the estimated tree size is below the resource limit.

```
procedure UpdateNewNode(n,G)
  ; AN(n,d) – Actual number of nodes to depth d
  ; AG(n,d) – Actual number of goals to depth d
  For d from 1 to D
    AN(n,d) = 1
    If G(n) then AG(n,d) = 1
    else AG(n,d) = 0
  UpdateParents(n, 1, G(n))

procedure UpdateParents(n,Depth,GoalFlag)
  if Depth < D
    For d from Depth to D
      AN(p,d) ← AN(p,d) + 1
      if GoalFlag then GN(p,d) ← GN(p,d) + 1
    For p ∈ parents(n)
      UpdateParents(p, Depth + 1, GoalFlag)
```

Figure 3: An algorithm for updating the partial yield

```
procedure Yield(n,d)
  C ← {s ∈ Childs(n) | AN(s, d − 1) > Threshold}
  if |C| > 0 then
    Return Avg({GN(c,d−1)/AN(c,d−1) | c ∈ C})
  else Return
    Avg({Yield(p, Min(d + 1, D)) | p ∈ Parents(n)})
```

Figure 4: An algorithm for yield estimation

The algorithms described above assume that each goal node has the same value. The scheme can be easily generalized to the case where each goal node carries different value. In such a case, the node weight should be added to the counter instead of 1.

Generalizing yield

One problem with the above approach is that partial yield information allows us to predict yield only to nodes that are near-by in the graph structure. If we can extract a vector of domain-dependent features for each state (node), we can use induction algorithms to infer yield based on states that are near-by in the feature space. Examples will be supported nodes (with sufficiently large subtrees). The examples will be tagged by their partial yield. We can then apply common induction algorithms to distinguish between states with low and high yield, or, more generally, to infer the yield function. Since learning is performed online, we are interested in reducing both learning cost and classification (prediction) cost. For reducing learning costs we can use *lazy* learners such as KNN. These algorithms, however, carry high prediction cost which can be somewhat reduced by indexing. Decision tree algorithms have the advantage of low-cost prediction but high learning cost. This can be somehow reduced by enlarging the time gap between learning sessions.

Applying our framework to the WWW domain

One of the main motivations for this research is the problem of *focused crawling* in the web (Chakrabarti, van den Berg, & Dom 1999; Cho & Garcia-Molina 2000; Menczer *et al.* 2001). The task is to find as many goal pages as possible using limited resources where the basic resource unit is usually retrieving a page from a link. Previous work on focused crawling concentrated on web-specific techniques for directing the search. We believe that our generalization of single-goal heuristic search to multiple-goal search can contribute to the task of focused crawling. In this section we discuss various issues related to implementing the above framework to the web domain.

- There are several ways to define goal web pages.
  - Boolean keywords query – this can be extended to word in a specific tag environment such as titles or meta-tags.
  - Parametric query – request pages that satisfy some parameter specifications. A known example for such a parameter is the page-rank (Najork & Wiener 2001).
  - Query by example – a set of pages tagged as goals or none-goals. These pages can be used to build a classifier for recognizing goals.
- When using a keyword-based goal predicate, we can use the keywords also for the multiple-goal heuristics for indicating distance to goals. The vector of keywords can also serve to produce a vector of heuristic distances needed for the *sum* and similar accumulating heuristics. In addition, one can use specific domain-independent keywords such as "index" or "collection" to indicate good hubs (Kleinberg 1999; Borodin *et al.* 2001; Cohen & Fan 1999).
- When describing the web as a search graph, web pages are considered as states and links can be considered as operators. Thus, in addition to using page-heuristics to evaluate a particular page (state), we can also use link-heuristics to evaluate the application of a specific operator in the given state. Such heuristics can use, for example, the text around the link and the link itself.
- When generalizing over web pages for estimating yield, we can use any of the known schemes for extracting features of web pages. For example, we can use words with high information content measured by various methods such as TFIDF (Joachims 1997).

Empirical evaluation

In this section we show an empirical evaluation of the multiple-goal search methods described above.

Experimental methodology

We test our algorithm by running it until it searches the whole graph and counting the number of goals it collects. This gives us the performance profile of the algorithm which is a general indicator for its expected performance. Our experiments are conducted in the context of two domains. Most of our experiments are in the web domain. Performing rigorous empirical research on the web is problematic. First, the web is dynamic and therefore may be changed between different runs of the algorithms. Second, crawling in the web require enormous time which disallow parametric experimentation. To solve the above problems we downloaded a small section of the web to local storage and performed the experiments using the local copy (Hirai *et al.* 2000). Specifically, we downloaded the domain `stanford.edu` which contains, after some cleanup, approximately 350000 valid and accessible HTML pages.

To show the generality of our approach we also run our algorithms on the task of planing object collection in a simulated grid environment of 50×50 with random "walls" inserted as obstacles. There are parameters controlling the maximal length of walls and desired density of the grid. Walls are inserted randomly making sure that the resulting graph remains connected.

Basic experiment in the Grid domain

Our basic experiment involves running the multiple-goal best-first algorithm with the *sum* heuristic with and without front-advancement. We compared their performance to a multiple-goal version of BFS and multiple-goal best-first with traditional Manhattan distance heuristic. Figure 5

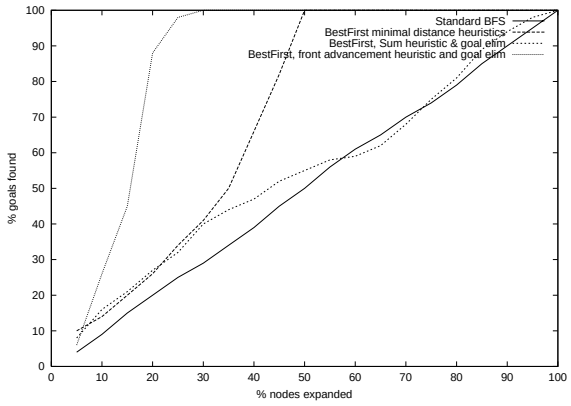

Figure 5: Working with several algorithms on grid domain.

shows the results obtained. Each point in the graphs is an average of 100 runs. For each run a new random grid was generated and a random set of goal states was selected. As expected the regular heuristic method is better than BFS and the front-advancement method is much better than the others. If we are limited, for example, to generate 20% of the total nodes, this method finds 3.5 times as many goals as the other methods. It is a bit surprising that the *sum* heuristic performed miserably. This is probably due to its attempt to progress towards all goals simultaneously.

Experiments in the WWW domain

We defined 8 goal predicates relevant to the Stanford domain and tagged the pages according to the defined predicates. Those goal sets included the set of all personal home pages, the set of pages about robotics and others. For each of the goal predicate, we run BFS and multiple-goal best-first search with goal-specific heuristic based on the goal keywords and general "hub" keywords as described in the

previous section. Figure 6 shows the results obtained for

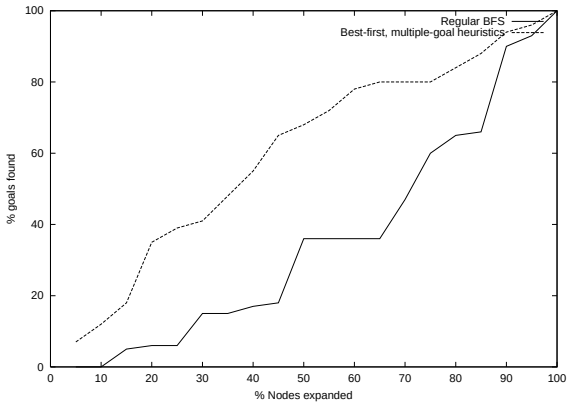

Figure 6: Multiple-goal best-first applied to the web domain

the "robotics" goal predicates. We can see that the heuristic method shows much better performance than the blind method. For example, if allocated a number of page retrievals which is 30% of the total pages, the heuristic method retrieves 3 times the goals retrieved by the blind method. Similar results were obtained for the other goal predicates.

Using front advancement in the WWW domain

We tested the front advancement method in the web domain for the "home pages" goal predicate restricted for a specific set of 100 people. We obtained a set of distance heuristics, one for each person. The functions are based on a summary table, found in the domain, which contains a short information about each person. We used these 100 heuristics for multiple-goal heuristic implementing the front-advancement method. Figure 7 shows the results obtained compared to

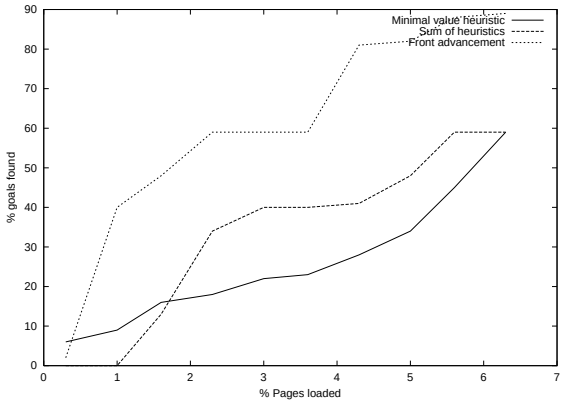

Figure 7: The effect of front advancement

the *sum* and minimal-value heuristics. We can see that indeed the *sum* heuristic is better than the simple minimizing heuristic but the front advancement is superior to both.

Learning yield during search progress

We tested both of our learning methods in the web domain. For generalizing yield we have used nearest-neighbor classification. Figure 8 shows the results obtained. We can see that

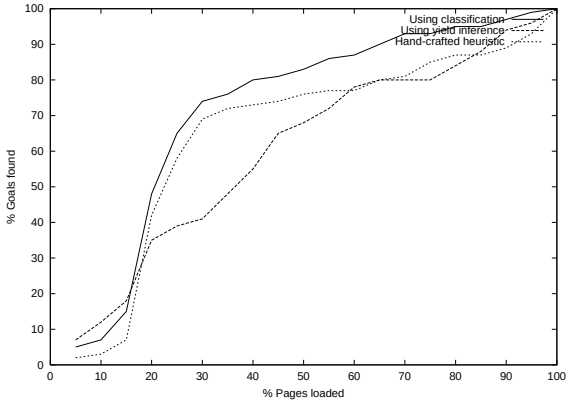

Figure 8: Learning yield in the web domain

using yield inference leads to better performance than using a hand-crafted heuristic. Using the generalizing method produced even better results.

Combining heuristic and yield methods

If we look carefully at the graphs of Figure 8 we can see that at the initial stages of the search the heuristic method performs better than the adaptive methods. The reason is that the adaptive methods require sufficient examples to generate predictions. We decided to try a combined method which uses a linear combination of the yield prediction and the heuristic estimation. Figure 9 shows the results ob-

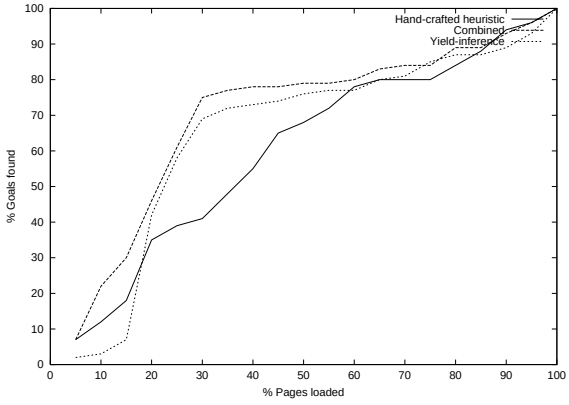

Figure 9: Combining yield-inference with heuristic search

tained for the yield-inference method, the heuristic method and the combined algorithm. We can see that indeed the combined method is better than both algorithms. An interesting phenomenon is that the combined method improves over *both* methods (rather than combining the maximum of both). The reason is that very early in the search process the

hand-crafted heuristic leads to enough goals to jump-start the learning process much earlier.

Contract algorithm versus anytime algorithm

In the previous subsections we have shown results for multiple-goal search behaving as an anytime algorithm. In practice we expect to use the search methods as *contract* algorithms (Russell & Zilberstein 1991) where the allocation of resources (or the requirement for the number of goals) are given as input to the algorithm. Contract algorithms can use the resource allocation input to improve their performance versus anytime algorithms. To obtain such improvement we can use the ability of the yield estimation algorithm to take into account the resource allocation or the number of goals requirement. In Table 1 we show the performance of our yield-inference algorithm when given a resource allocation (given by the percentage of page retrievals out of the total number of pages) for the two versions: the version that takes into account the limit and the version that does not. We can

Resource allocation	%goals found	
	Contract	Anytime
5%	7%	19%
10%	22%	33%
20%	46%	59%
50%	79%	83%

Table 1: Contract vs. anytime performance.

see that indeed using the allocation as input improves the performance of the algorithm by up to 250%. Similar results were obtained when the algorithm is limited by the number of goals instead of the number of page retrievals.

The results reported in this section are based on the assumption that the dominant resource is page retrieval over the net. There is a small overhead associated with maintaining the statistics for each node. The cost of learning can be high, but can be reduced by evoking the induction process less frequently. The space complexity is linear in the number of visited pages. If this number becomes too high, we can use one of the methods developed for memory-bounded heuristic search.

Conclusions

The work described in this paper presents a new framework for heuristic search. In this framework the task is to collect as many goals as possible within the allocated resources. We showed the inadequacy of traditional heuristics to this type of tasks and present heuristics that are more appropriate for multiple-goal search. In particular we introduced the *yield* heuristic that attempts to estimate the cost and the benefit of exploring a subtree below a search node. We also introduced two methods for online learning of the yield heuristic. One which is based on the partial yield of brother nodes and another which generalizes over the state feature space. We applied our methodology to the task of selective web crawling and showed its merit under various conditions. Our framework is not limited to the web domain. We showed an example of applying it to the task of planing object collection in a simulated grid environment. Our framework is applicable for a wide variety of problems where we are interested in finding many goals states instead of only one. For example, in the context of constraint satisfaction it may be useful to find many solutions and apply another algorithm for selecting between the found set.

Acknowledgments

We would like to thank Adam Darlo, Irena Koifman and Yaron Goren who helped us in programming. This research was supported by the fund for the promotion of research at the Technion and by the Israeli Ministry of Science.

References

Borodin, A.; Roberts, G. O.; Rosenthal, J. S.; and Tsaparas, P. 2001. Finding authorities and hubs from link structures on the world wide web. In *The 10th international WWW conference*, 415–429. ACM Press.

Brin, S., and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems* 30(1–7):107–117.

Chakrabarti, S.; van den Berg, M.; and Dom, B. 1999. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks* 31(11–16):1623–1640.

Cho, J., and Garcia-Molina, H. 2000. The evolution of the Web and implications for an incremental crawler. In El Abbadi, A.; Brodie, M. L.; Chakravarthy, S.; Dayal, U.; Kamel, N.; Schlageter, G.; and Whang, K.-Y., eds., *VLDB 2000*, 200–209. Los Altos, CA 94022, USA: Morgan Kaufmann Publishers.

Cohen, W. W., and Fan, W. 1999. Learning page-independent heuristics for extracting data from Web pages. *Computer Networks* 31(11–16):1641–1652.

Hirai, J.; Raghavan, S.; Garcia-Molina, H.; and Paepcke, A. 2000. Webbase: A repository of web pages. In *Proceedings of the 9th International WWW Conference*, 277–293.

Joachims, T. 1997. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proc. 14th International Conference on Machine Learning*, 143–151. Morgan Kaufmann.

Kleinberg, J. M. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM* 46(5):604–632.

Kumar, R.; Raghavan, P.; Rajagopalan, S.; Sivakumar, D.; Tomkins, A.; and Upfal, E. 2000. The Web as a graph. In *Proc. 19th Symp. Principles of Database Systems, PODS*, 1–10. ACM Press.

Menczer, F.; Pant, G.; Srinivasan, P.; and Ruiz, M. 2001. Evaluating Topic-Driven web crawlers. In *Proceedings of SIGIR-01)*, 241–249. New York: ACM Press.

Najork, M., and Wiener, J. L. 2001. Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th International World-Wide Web Conference*, 114–118.

Russell, S. J., and Zilberstein, S. 1991. Composing real-time systems. In *Proceedings of IJCAI-91*. Sydney: Morgan Kaufmann.