Journal of Artificial Intelligence Research 26 (2006) 417-451 Submitted 11/05; published 08/06

Multiple-Goal Heuristic Search

Dmitry Davidov DMITRY@CS.TECHNION.AC.IL
Shaul Markovitch SHAULM@CS.TECHNION.AC.IL
Computer Science Department
Technion, Haifa 32000, Israel

Abstract

This paper presents a new framework for anytime heuristic search where the task is to achieve as many goals as possible within the allocated resources. We show the inadequacy of traditional distance-estimation heuristics for tasks of this type and present alternative heuristics that are more appropriate for multiple-goal search. In particular, we introduce the marginal-utility heuristic, which estimates the cost and the benefit of exploring a subtree below a search node. We developed two methods for online learning of the marginal-utility heuristic. One is based on local similarity of the partial marginal utility of sibling nodes, and the other generalizes marginal-utility over the state feature space. We apply our adaptive and non-adaptive multiple-goal search algorithms to several problems, including focused crawling, and show their superiority over existing methods.

1. Introduction

Internet 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; Douglis, Feldmann, Krishnamurthy, & Mogul, 1997), and they still leave a large part of the Web uncovered (Lawrence & Giles, 1998; Najork & Wiener, 1998). Many times, however, it is necessary to retrieve only a small portion of Web pages dealing with a specific topic or satisfying various user criteria. Using brute-force crawlers for such a task would require enormous resources, most of which would be wasted on irrelevant pages. Is it possible to design a *focused* crawler that would scan only relevant parts of the Web and retrieve the desired pages using far fewer resources than the exhaustive crawlers?

Since the Web can be viewed as a large graph (Cooper & Frieze, 2002; Kumar, Raghavan, Rajagopalan, Sivakumar, Tomkins, & Upfal, 2000; Pandurangan, Raghavan, & Upfal, 2002), where pages are nodes and links are arcs, we may look for a solution to the above problem 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 inappropriate for focused crawling, which uses an entirely different setup. The crucial difference between heuristic search and focused crawling is the success criterion. 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 focused crawling setup, the search continues to reach as many goal states as possible within the given resources.

Changing the success criterion of existing search algorithms is not enough. Most informed search algorithms are based on a heuristic function that estimates the distance from a node to the nearest goal node. Such heuristics are usually not appropriate for multiple-goal search. Consider the search graph described in Figure 1. The grey area is the expanded

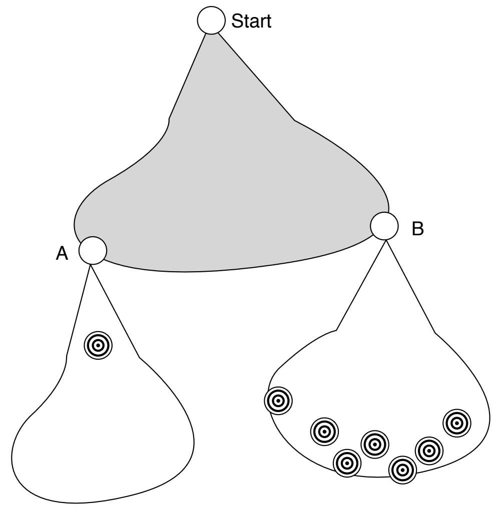


Figure 1: Using a distance-estimation heuristic for a multiple-goal search problem

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

For many problems in a wide variety of domains where we are interested in finding a set of goals rather than a single goal. In genetic engineering, for example, we want to find multiple possible alignments of several DNA sequences (Yoshizumi, Miura, & Ishida, 2000; Korf & Zhang, 2000). In chemistry we may want to find multiple substructures of a complex molecule. In robotics, we may want to plan paths for multiple robots to access multiple objects. In some of these cases, a possible solution would be to invoke single-goal search multiple times. Such an approach, however, is likely to be wasteful, and for resource-bounded computation, we may wish to exploit the multiple-goal[1] nature of the problem to make the search more efficient.

The specific multiple-goal task of focused crawling has received much attention (Chakrabarti, van den Berg, & Dom, 1999; Cho & Garcia-Molina, 2000; Cho, García-Molina, & Page, 1998; Diligenti, Coetzee, Lawrence, Giles, & Gori, 2000; Rennie & McCallum, 1999) because of the popularity of the Web domain. Most of these works, however, focused on Web-specific techniques tailored for the particular problem of crawling.

The goal of the research described in this paper is to establish a new domain-independent framework for multiple-goal search problems and develop anytime heuristic algorithms for solving them efficiently. Our framework focuses mainly on problem domains where we are looking for the goal states and the paths to the goals either are irrelevant, or their cost is of no concern (except for its effect on the search cost).

1. Note that the term "multiple-goal" is also used in the planning domain. There, however, the task is to satisfy a set of dependent goals with possible order constraints.

We start with a formal definition of the multiple-goal search problem. We then describe versions of existing heuristic search algorithms, modified for the multiple-goal framework. The main differences between single-goal and multiple-goal search are the heuristic functions used. We describe a new set of heuristics that are better suited for multiple-goal search. In particular, we introduce the marginal-utility heuristic, which considers the expected costs as well as the expected benefits associated with each search direction. It is difficult to specify such heuristics explicitly. We therefore present adaptive methods that allow online learning of them. Finally we describe an extensive empirical study of our algorithms in various domains, including the focused crawling problem.

The contributions of this paper are fourfold:

1. We identify and define the framework of multiple-goal heuristic search. This is a novel framework for heuristic search.

2. We define a set of search algorithms and heuristics that are appropriate for multiple-goal search problems.

3. We define a utility-based heuristic and present a method for automatic acquisition of it via online learning.

4. We provide extensive empirical study of the presented methods in various domains and show their superiority over existing general algorithms.

2. The Multiple-Goal Search Problem

Let $\langle S, E \rangle$ be a potentially infinite state graph with finite degree, where S is a set of states and $E \subseteq S \times S$ is a set of edges. The *single-goal search problem* can be defined as follows:

1. *Input:*
   - A set of initial states $S_i \subseteq S$
   - A successor function $\text{Succ} : S \rightarrow 2^S$ such that $\text{Succ}(s) = \{s' \mid \langle s, s' \rangle \in E\}$. (Sometimes Succ is given implicitly be a finite set of operators O.)
   - A goal predicate $G : S \rightarrow \{0, 1\}$. We denote $S_g = \{s \in S \mid G(s)\}$. Sometimes S_g is given explicitly.

2. *Search objective:* Find a goal state $g \in S_g$ such that there is a directed path in $\langle S, E \rangle$ from a state in S_i to g. Sometimes we are also interested in the path itself.

3. *Performance evaluation:* Although the performance evaluation criterion is not commonly considered to be an integral part of the problem definition, we do consider it as such. This is because it determines the class of algorithms to be considered. The most common criteria for evaluating a search are the solution quality, usually measured by the solution path cost, and search efficiency, mostly evaluated by the resources consumed during the search.

Although the *multiple-goal search problem* bears some similarity to the single-goal search problem, it differs in several ways:

1. *Input:* Includes an additional resource limit R. For simplicity of presentation we assume that R is given to us as a number of generated nodes[2] . Later we will discuss this assumption.

2. *Search objective:* Find a *set* of goal states $S_g^R \subseteq S_g$ which satisfies:

   - For each $s \in S_g^R$, there is a directed path in $\langle S, E \rangle$ from some $s_i \in S_i$ to s.
   - The search resources consumed should not exceed R.

   Sometimes we are also interested in the set of corresponding paths.

3. *Performance evaluation:* $\left|S_g^R\right|$. Obviously, higher values are considered better.

While it looks as if we mix here reasoning with meta reasoning by inserting resource limit as part of problem input, many problems are much more naturally defined with such a resource limitation. Consider, for example, the minimax algorithm, where the maximal depth of search (which determines the resources consumed) is given as part of the algorithm's input. The above formulation, where the resource limit is given as an input, falls under the scope of problems solved by *anytime algorithms* (Boddy & Dean, 1994; Hovitz, 1990; Zilberstein, 1996) and more specifically by *contract algorithms* (Russell & Zilberstein, 1991; Zilberstein, Charpillet, & Chassaing, 1999).

Sometimes the resource limit is not known in advance. In such a setup, the search algorithm can be interrupted at any time and required to return the current set of collected goals. This type of problem is solved by *interruptible anytime algorithms* (Hansen & Zilberstein, 1996; Russell & Zilberstein, 1991). An alternative formalization is to require the algorithm to find a specified number of goals and to evaluate its performance by the resources consumed during the search.

3. Multiple-Goal Heuristic Search Algorithms

Assume that $h_{mg} : S \rightarrow \Re$ is a heuristic function that estimates the merit of states with respect to the objective and performance evaluation criterion of a multiple-goal search problem as defined in the previous section.

Our objective is to develop search algorithms that can exploit such heuristics in a similar manner to the heuristic search algorithms developed for single-goal search problems. We start by describing the multiple-goal version of greedy best-first search[3].

There are two main differences between the existing single-goal best-first search algorithm and our newly defined multiple-goal version:

1. While single-goal best-first search stops as soon as it encounters a goal state, the multi-goal version collects the goal and continues until the allocated resources are exhausted.

2. Our framework is therefore applicable to any resource that is proportional to the number of generated nodes. That can be CPU time, internet bandwidth, energy consumption by robots, etc. For best-first search algorithms, the number also corresponds to the memory consumption.
3. We follow Russell and Norvig (2003, page 95) who use this term to describe a search algorithm that always expands the node estimated to be closest to a goal.

2. While single-goal best-first search typically uses a heuristic function that tries to approximate the distance to the nearest goal, the multiple-goal version should use a different type of heuristic that is more appropriate for the multiple-goal task.

Most of the other heuristic search algorithms can also be modified to find multiple goals. The A^*_ϵ algorithm by Pearl and Kim (1982) can be converted to handle multiple goal search by collecting each goal it finds in its *focal* list. These are goals with $\epsilon-$optimal paths. A multiple-goal heuristic can be used to select a node from the *focal* list for expansion. The algorithm stops, as before, when the allocated resources are exhausted. In addition, we can stop the algorithm when all the nodes in *focal* satisfy $f(n) > (1+\epsilon)g_{min}$, where g_{min} is the minimal g value among the collected goals.

Multiple-goal hill-climbing uses a multiple-goal heuristic to choose the best direction. We modify this algorithm to allow the search to continue after a goal is found. One possible method for continuing the search is to perform a random walk from the found goal.

Multiple-goal backtracking works similarly to the single goal version. However, when a goal is encountered, the algorithm simulates failure and therefore continues. A multiple goal heuristic can be used for ordering the operators in each node. For constraint satisfaction, that means ordering the values associated with a variable.

4. Heuristics For Multiple-Goal Problems

In the introduction, we illustrated the problem of using a traditional distance-estimation heuristic for multiple-goal search. One of the main problems with using such a distance-estimation heuristic is that it does not take into account goal density 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, rather than the much more fruitful right branch. In this section we consider several alternative heuristic functions that are more appropriate for multiple-goal search.

4.1 The Perfect Heuristic

Before we describe and analyze heuristic functions for multiple-goal search, we would like to consider the function that we are trying to approximate. Assume, for example, that we perform multiple-goal greedy best-first search and that we have perfect knowledge of the search graph. What node would we like our heuristic to select next? Assume that the given resource limit allows us to expand additional M nodes, and look at all the search forests[4] of size M rooted at the current list of open nodes. A perfect heuristic will select a node belonging to the forest with the largest number of goals.

Definition 1 *Let $S_{open} \subseteq S$ be the set of currently open states. Let R be the resource limit and R_c be the resources consumed so far. Let $S_{gf} \subseteq S_g$ be the set of goals found so far. Let F be a set of all possible forests of size $R - R_c$ starting from roots in S_{open}. A forest $f \in F$ is* optimal *if and only if*

$$\forall f' \in F, \left|(S_g \setminus S_{gf}) \cap f'\right| \leq \left|(S_g \setminus S_{gf}) \cap f\right|.$$

4. While the search space is a graph, the search algorithm expands a forest under the currently open nodes.

A state $s \in S_{open}$ is optimal*, denoted as $OPT(s)$, if and only if there exists an optimal forest f such that $s \in f$.*

Definition 2 *A heuristic function h is* perfect *with respect to a multiple-goal search problem if for every possible search stage defined by S_{open},*

$$\forall s_1, s_2 \in S_{open} \left[\mathrm{OPT}(s_1) \wedge \neg\mathrm{OPT}(s_2) \Longrightarrow h(s_1) < h(s_2)\right].$$

Thus a perfect heuristic never selects for expansion a state that is not in an optimal forest. Using such a heuristic in multiple-goal best-first search will make the search optimal. Note that the optimality is with respect to search resources and not with respect to the cost of paths leading to the goal states.

Obviously, the above definition does not lead to a practical multiple-goal heuristic. Even for simple problems, and even when we have perfect knowledge about the graph, calculating such a heuristic is very hard. This is because the number of possible forests is exponential in the resource limit and in the number of nodes of the open list.

4.2 Sum Heuristics

Many search algorithms that look for a single goal state use a heuristic function that estimates the cost of the cheapest path to a goal state. Optimizing algorithms such as A^* require admissible heuristics (that underestimate the real distance to the goal) while *satisficing* search algorithms, such as greedy best-first, can use non-admissible heuristics as well. Distance-estimation heuristics were therefore developed for many domains. In addition, several researchers have developed automatic methods for inferring admissible heuristics by relaxation (Held & Karp, 1970; Mostow & Prieditis, 1989; Prieditis, 1993) and by pattern databases (Culberson & Schaeffer, 1998; Gasser, 1995; Korf & Felner, 2002).

Figure 1 illustrates that a straightforward use of distance heuristics for multiple-goal search is not appropriate. Here we define a method for utilizing (admissible or non-admissible) distance heuristics for multiple-goal search. Assume that the set of goal states, S_g, is given explicitly, and that we are given a common distance-estimation heuristic $h_{dist}(s_1, s_2)$ that estimates the graph distance between two given states[5]. The *sum-of-distances* heuristic, denoted as h_{sum}, estimates the sum of distances to the goal set:

$$h_{sum}(s) = \sum_{g \in S_g} h_{dist}(s, g). \tag{1}$$

Minimizing this heuristic will bias the search towards larger groups of goals, thus selecting node B in the example of Figure 1. This is indeed a better decision, provided that enough resources are left for reaching the goals in the subgraph below B.

4.3 Progress Heuristics

One problem with the *sum* heuristic is its tendency to try to progress towards all of the goals simultaneously. Hence, 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

5. Assume that the Euclidean distance in the figure reflects the heuristic distance.

the distances and increases others, leading to a more-or-less constant sum. In such constant-sum regions, an algorithm which relies only on the sum heuristic will have no information about which node to choose for expansion. Even when there are a few distinct groups of

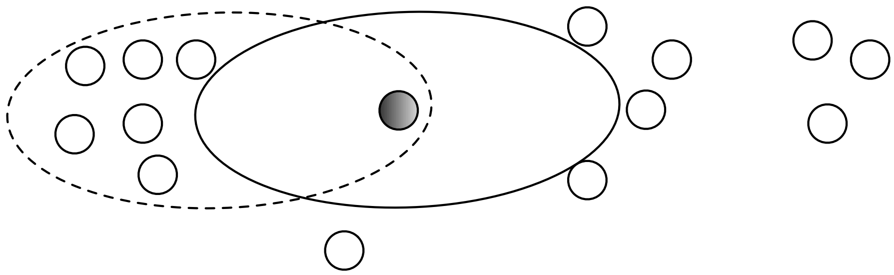


Figure 2: The behavior of the sum heuristic vs. that of the progress heuristic. The solid-line ellipse indicates the area covered by a search using the sum heuristic. The dotted-line ellipse marks the area searched using the progress heuristic.

goals in different directions, the sum heuristic may lead to simultaneous progress towards all the groups. The two groups of goal states shown in Figure 2 illustrate this problem. The sum heuristic strategy will work only if there are enough enough resources to reach all the groups. If, however, the resources are not sufficient, the sum heuristic may waste all the available resources in trying to progress towards all the groups, but reaching none.

To avoid such problems, we define the *progress* heuristic, which takes into account the number of goals towards which progress is made and the average distance to them. Thus, instead of trying to pursue multiple groups of goals, this heuristic will pursue one group at a time, preferring less distant groups.

Let S_{open} be the set of currently opened states. As before, we assume that we have the explicit goal list, $S_g = \langle g_1, \ldots, g_k \rangle$, and a distance-estimation heuristic, h_{dist}. Let $m_i = \min_{s \in S_{open}} h_{dist}(s, g_i)$ be the minimal estimated distance from the search frontier to the goal g_i. For each $s \in S_{open}$ we define $G_p(s) = \{g_i \in S_g \mid h_{dist}(s, g_i) = m_i\}$ to be the set of goals for which s is estimated to be the closest among the states in the search frontier. The average estimated distance between s and the states in G_p is $D_p(s) = \frac{\sum_{g \in G_p(s)} h_{dist}(s,g)}{|G_p(s)|}$ be their average distance from s. We are interested in states that have many members in G_p with small average distance. Hence, we define the *progress* heuristic to be

$$h_{progress}(s) = \frac{D_p(s)}{|G_p(s)|}. \tag{2}$$

Minimizing this heuristic will direct the search to larger and closer groups of goals towards which progress is made. For example, in the simple space shown in Figure 2, the progress heuristic will advance correctly, towards the group at the left side as indicated by the dashed ellipse. Note that although the right group is larger, the progress heuristic nonetheless prefers the left one because of its smaller distance. Since the progress heuristic considers each goal exactly once, it is not misled by multiple paths to the same goal.

4.4 Marginal-Utility Heuristics

When comparing two search directions, we have so far considered only the concentration of goal states, preferring directions that lead to larger groups of goals. Thus, the former heuristics would have considered node A and node B in Figure 3 to be equivalent. This reasoning, however, does not take into account the resources invested to reach the set of goals. In the example of Figure 3, it is clear that visiting the set of goals under node B requires less search resources than those under node A. Therefore node B is preferable.

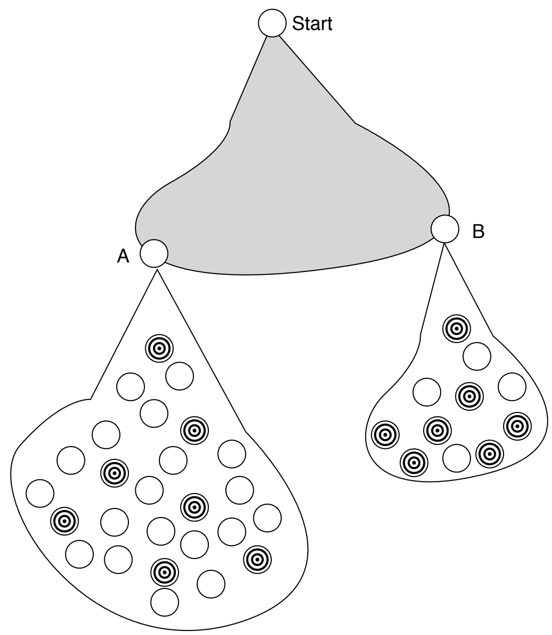


Figure 3: Searching from node A will result in the same number of goals as searching from node B. It will consume, however, far greater resources.

We account for these cases by suggesting another approach for multiple-goal heuristics, one that considers not only the expected *benefit* of the search but also the expected *cost*. Obviously, we prefer subgraphs where the cost is low and the benefit is high. In other words, we would like a high return for our resource investment. We call a heuristic that tries to estimate this return a *marginal-utility* heuristic.

Assume (for now) that the search space $\langle S, E \rangle$ is a tree. Let $T(s)$ be the set of all states that are reachable from s. Let $T_g(s) = T(s) \cap S_g$ be the set of all goal states reachable from s. Let $S_v \subseteq S$ be the set of all states visited during a completed search process. We define the *marginal utility* of state $s \in S_v$ with respect to S_v as

$$\mathrm{MU}(s) = \frac{|T_g(s) \cap S_v|}{|T(s) \cap S_v|}. \tag{3}$$

Thus, MU measures the density of goal states in the subtree expanded by the search process.

One possible good search strategy is to select states that are should eventually yield high values of MU with respect to S_v. If the search process has consumed $R_c \leq R$ resources so far, then the largest tree that can be visited is of size $r = R - R_c$. In Section 4.1 we define the perfect heuristic by considering all the possible ways of distributing r among the open nodes. This approach is obviously impractical, and we will take here a greedy approach instead. We look for a heuristic function $h_{MU}(s, r)$ that tries to estimate the best marginal utility of s, assuming that all the remaining resources, r, are consumed while exploring $T(s)$. Let $T(s, r)$ be the set of all trees of size r under root s. $h_{MU}(s, r)$ tries to estimate the *resource-bounded marginal utility*, defined as

$$\mathrm{MU}(s, r) = \max_{T \in T(s,r)} \frac{|T_g(s) \cap T|}{r}. \tag{4}$$

$\mathrm{MU}(s, r)$ measures the best ratio between the number of goals achieved and the search resources used for it. Naturally, it is very difficult to build heuristics that estimate marginal utility accurately. In the following sections we show how such heuristics can be learned.

4.5 Additional Considerations

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, where it would have been preferable for the search to progress towards undiscovered goals. If an explicit set of goal states is given - as it is for the sum and progress heuristics - we can disable the influence of visited goals by simply removing them from the set. If a set of features over states is given instead, we can reduce the effect of visited goals by preferring nodes that are farther from them in the feature space. Specifically, let $d(n)$ be the minimal distance of n from members in the set of visited goals, and let $h(n)$ be the multiple-goal heuristic value of n. The modified heuristic will be $h'(n) = h(n)(1+c_1 e^{-c_2 d(n)})$ where c_1 and c_2 are parameters that determine the magnitude of the effect of $d(n)$. The second term is the penalty we add to the heuristic value. This penalty decays exponentially with the distance from the visited goals.

Note that there is a tension between the tendency to search dense groups of goals and and the tendency to push the search away from visited goals. When the groups of goals are dense, the above method can be detrimental because finding some of the goals in a group reduces the tendency to pursue the other goals of the same group. This effect can be controlled by the c_i parameters. For domains with high goal density, c_i should be set to lower values. Hence, these values can be set dynamically during the search, according to measurements of goal density in the explored graph.

One problem with using the marginal utility heuristic in non-tree graphs is the possible overlap of marginal utility. That means that our search algorithm might pursue the same set of goals from different directions. One way to overcome this problem is to try to diversify the search progress by measuring the feature-based average distance between the "best" nodes and the set of recently expanded nodes, and prefer those with maximal diversity from the nodes explored. This gives maximal diversity in exploration directions and should minimize the expected overlap of visited subtrees.

5. Learning Marginal Utility Heuristics

While it is quite possible that marginal-utility heuristics will be supplied by the user, in many domains such heuristics are very difficult to design. We can use a learning approach to acquire such marginal-utility heuristics online during the search. We present here two alternative methods for inferring marginal utility. One approach estimates the marginal utility of a node based on the partial marginal utility of its siblings. The other approach predicts the marginal utility using feature-based induction.

5.1 Inferring Marginal Utility Based on Marginal Utility of Siblings

The first approach for predicting marginal utility is based on the assumption that sibling nodes have similar marginal utility. We define the *partial* marginal utility of a state s at step t of executing a multiple-goal search algorithm as the number of goals found so far in the subtree below s divided by the number of states visited so far in this subtree. Thus, if $S_v(t)$ is the set of states visited up to step t, then the partial marginal utility is defined as

$$MU(s,t) = \frac{|T_g(s) \cap S_v(t)|}{|T(s) \cap S_v(t)|}. \tag{5}$$

The method estimates the marginal utility of the siblings based on their partial marginal utility, and the marginal utility of the node based on the average estimated marginal utility of the siblings.

As discussed in the previous subsection, the expected marginal utility of a node strongly depends on the resources invested in exploring it. Thus, to learn the heuristic $h_{MU}(s,r)$, one would need to estimate partial marginal utility values for different values of r. One way of reducing the complexity of this two-dimensional estimation is to divide it into two stages: estimating the depth of the tree under s that is searchable within r resources, and estimating the marginal utility for the predicted depth. Thus, if $\widetilde{MU}_{depth}(s,d)$ is the estimated marginal utility when searching node s to depth d, we compute the estimated marginal utility of a node s using r resources by $\widetilde{MU}_{resources}(s,r) = \widetilde{MU}_{depth}(s, \widetilde{d}(s,r))$, where $\widetilde{d}(s,r)$ is the estimated depth when searching under node s using r resources. In the following subsections we show how these values are estimated.

5.1.1 Updating Partial Marginal-Utility Values

We maintain for each node two vectors of D counters, where D is a parameter that limits the maximal lookahead of the partial marginal utility. One vector, $N(n)$, stores the current number of visited nodes for node n for depth $1,\ldots,D$, where $N_i(n)$ contains the current number of visited nodes under n with depth of less or equal to i. The other vector, $G(n)$ holds similarly the number of visited goals.

Whenever a new node n is generated, the appropriate entries of its ancestors' N vectors are incremented. If n is a goal, then the G vectors are updated as well. If p is an ancestor of n connected to it with a path of length $l \leq D$, then $N_l(p), \ldots N_D(p)$ are incremented by one. If more than one path exists between n and p, we consider only the shortest one. The memory requirements for this procedure are linear in D and in the number of stored nodes.

The number of operations required for one marginal-utility update is bounded by $O(B_{degree}^D)$, where B_{degree} is the upper bound on the maximum indegree in the graph. Therefore, if the backward degree is bounded, the number of calculations per node does not grow as the search progresses. The depth limit D determines the complexity of the update for a given search graph; hence, it is desirable to reduce its value. A value that is too low, however, will make it possible to infer only "local" marginal-utility values.

5.1.2 Inferring Marginal Utility

The inference algorithm estimates the marginal utility of a node on the basis of the average partial marginal utility of its siblings. Only nodes with sufficient statistics about partial marginal utility are used to predict the marginal utility of new nodes. We call such nodes *supported* nodes. If the node has no supported siblings, we base our estimate on the average estimated marginal utility of its parents (computed recursively using the same procedure). Figure 4 illustrates this method. In the left tree, the marginal utility of the grey node is

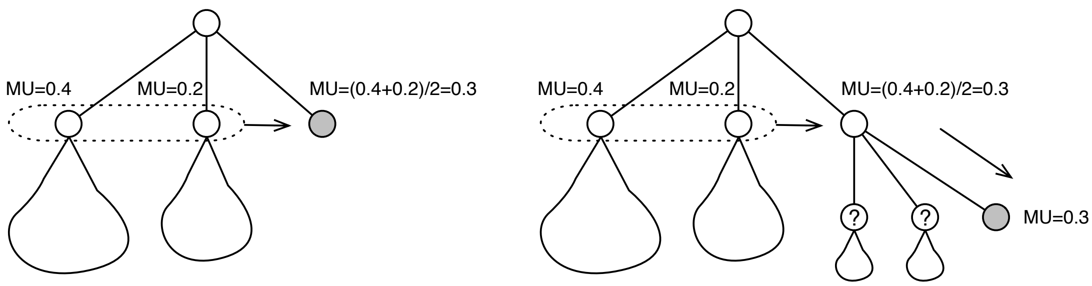


Figure 4: Inferring marginal utility from partial marginal utility of supported siblings (left) or supported uncles (right). Nodes with a question mark are unsupported.

computed as the average marginal utility of its siblings. In the right tree, the marginal utility of the grey node is computed as the average marginal utility of its uncles. The input to the marginal utility estimation heuristic is the remaining unconsumed resources, r. The algorithm first finds the largest depth, d, for which the number of predicted nodes is smaller than r. This prediction is based on the supported siblings or the parent of the node just as described above.

The found depth, d, is then used to determine which counters will be used to estimate the marginal utilities of the supported uncles. The complete algorithm is listed in Figure 5.

5.1.3 Sibling Clustering

In the algorithm described in Figure 5, the marginal utility of a node is induced by averaging the partial marginal utility of its siblings. If we can define a "meaningful" similarity metric between nodes, we can try making this prediction less noisy by using only the node's *similar* siblings. One way of doing so is to use the similarity metric to cluster the set of siblings and then generate a virtual node for each cluster. The virtual node is linked to the cluster members and its parent is the parent of the original sibling set. This is the only required change. The existing algorithm described in Figure 5 will do the rest. When predicting the marginal utility for a node, the algorithm first looks for the partial marginal utility

procedure MU(s,d)
  if Supported(s,d) **then return** $\frac{G_d(s)}{N_d(s)}$
  else $P \leftarrow$ Parents(s)
    if $|P| = 0$ **then return** 0
    $\mathit{SupportedSiblings} \leftarrow \{c \in \text{Children}(p) \mid p \in P, \text{Supported}(c)\}$
    if $|\mathit{SupportedSiblings}| > 0$ **then**
      return $\text{Avg}\left(\left\{ \frac{G_d(c)}{N_d(c)} \;\middle|\; c \in \mathit{SupportedSiblings} \right\}\right)$
    else return $\text{Avg}(\{MU(p, Min(d+1, D)) \mid p \in P\})$

procedure TreeSize(s,d)
  $P \leftarrow$ Parents(s)
  if Supported(s,d) **or** $|P| = 0$ **then return** $N_d(s)$
  else
    $\mathit{SupportedSiblings} \leftarrow \{c \in \text{Children}(p) \mid p \in P, \text{Supported}(c)\}$
    if $|\mathit{SupportedSiblings}| > 0$ **then**
      return $\text{Avg}(\{\{N_d(c) \mid c \in \mathit{SupportedSiblings}\})$
    else return $\text{Avg}\left(\left\{ \frac{\text{TreeSize}(p, \text{Min}(d+1, D))}{|\text{Children}(p)|} \;\middle|\; p \in P \right\}\right)$

procedure Get-marginal-utility(s,$\mathit{ResourceLimit}$)
  $\mathit{Depth} = max(d \leq D | \mathit{TreeSize}(s,d) < \mathit{ResourceLimit})$
  return MU(s, Depth)

Figure 5: An algorithm for marginal-utility estimation

of its siblings. In this case these are the members of its cluster. Only if these siblings are unsupported will the algorithm use the information from the other clusters propagated through the common parent.

This mechanism is illustrated in Figure 6. Without clustering, the predicted marginal utility of the unsupported nodes would have been the average of the three supported siblings, which is 0.5. Note that this average has a large variance associated with it. Clustering nodes A and B under one virtual node, and C, D, and E under another, yields (we hope) more accurate prediction, since it is based on more uniform sets. The similarity metric will usually be the Euclidean distance between vectors of features of the states corresponding to the sibling nodes. In some domains, we try to reduce the number of generated nodes by deciding at each step which node-operator pair to proceed with. In such cases we need a similarity measurement between operators to implement the above approach.

5.2 Feature-Based Induction of Marginal Utility

Unfortunately, partial marginal-utility information allows us to predict marginal utility only for nodes that are in proximity to one another in the graph structure. In addition, there

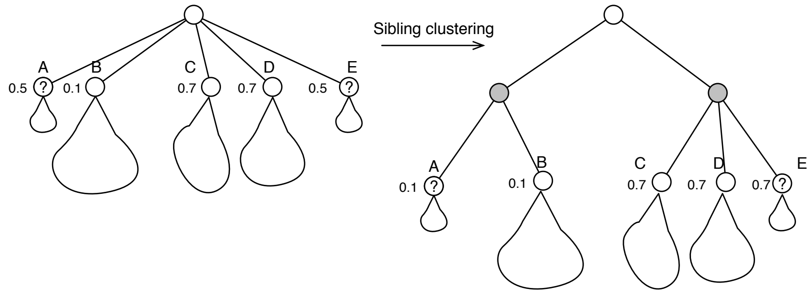


Figure 6: The effect of sibling clustering on marginal utility estimation

are domains where the local uniformity of sibling nodes with respect to marginal utility cannot be assumed. We can overcome these problems if we view marginal-utility inference as a problem of function learning and use common induction algorithms. For each depth d, we induce the marginal utility function using the set of supported nodes (with respect to d) as examples. The state features can be domain independent (such as the in-degree and out-degree of the node) or domain specific, supplied by the user. As for any induction problem, the quality of the induced function is highly dependent on the quality of the supplied features.

Since the above learning scheme is performed on-line, he high cost of learning, and of using the classifier directly, reduce the utility of the learning process. One way to essentially eliminate the learning costs is to use a *lazy* learner, such as KNN (Cover & Hart, 1967). This approach also has the advantage of being incremental: each new example contributes immediately to the learned model. The problem with this approach is the high costs associated with using the classifier.

An alternative approach would be to learn an efficient classifier such as a regression tree (Breiman, Friedman, Olshen, & Stone, 1984). This can be learned incrementally using algorithms such as ID5 (Utgoff, 1988). Batch learning algorithms such as C4.5 usually yield better classifiers than incremental algorithms. However, due to the higher cost of applying batch learning, one should decide how often to call it. Applying it after each node generation would increase the induction cost, but yield better classifiers earlier - which may improve the performance of the search process. Applying it at large intervals would reduce the induction costs but lead to poorer search performance.

The on-line learning process gives rise to another problem: the initial search period where there are not yet sufficient examples to make learning helpful. One way to reduce the effect of this lack of knowledge is by using classifiers that were induced beforehand, on-line or off-line, for goals that are similar to the goals of the current search.

6. Empirical Evaluation

To test the effectiveness of the methods described in the previous sections and to show their versatility, we experimented intensively on several domains. The most challenging domain, however, is *focused crawling* where we apply our algorithms to the task of collecting target web pages from a sub-web of millions of pages. We first compare the anytime behavior of our distance-based methods with that of uninformed search and best first search. Then we test the performance of our marginal utility methods. We also test the effect of the various suggested enhancements on the algorithms' performance. We also show realtime performance of our algorithm by allowing it to search the real web.

6.1 Experimental Methodology

We compare the performance of our algorithms to two competitors: breadth-first search and best-first search using distance estimation heuristics. Both algorithms were adopted to the multiple-goal framework by allowing them to continue the search after finding the first goal.

A basic experiment that compares two multiple-goal search algorithms is conducted in the following way:

1. A set of initial states and a goal predicate are given.
2. The algorithms perform multiple-goal search.
3. The resources consumed and the number of goals found during the execution are monitored.
4. The last two steps are repeated several times to accumulate sufficient statistics (all the algorithms contain at least one random component).
5. The performance of the two algorithms, measured by the number of goals found for the allocated resources, is compared.

The problem of estimating the performance of an algorithm when a resource allocation is given is most extensively discussed in the context of anytime algorithms. Measuring the performance of anytime algorithms is problematic (Hansen & Zilberstein, 1996). If a probability distribution over the resource allocation is given, then we can compute the expected performance of an anytime algorithm on the basis of its performance profile. In many cases, however, such a probability distribution is not available. We therefore measure the performance of the tested algorithm by means of the obtained quality for different resource allocation values. For multiple-goal search, the quality is measured by the number of goals found for the allocated resources. When we know the total number of goals, we report instead the percentage of goals found.

Alternatively, anytime algorithms can be evaluated by measuring the amount of resources required to achieve a given quality. For multiple-goal search, the most obvious measurement is time. Time, however, is overly affected by irrelevant factors such as hardware, software, and programming quality. Moreover, in the Web domain, most of it is spent accessing Web pages. How long this takes depends on many factors, such as network and server loads, which are irrelevant to our research topic.

We thus decided to measure resource consumption by the number of generated nodes. Nevertheless, we cannot ignore time completely: we must make sure that the overhead of the methods described in this paper does not outweigh their benefits. We therefore report time results for an experiment that uses the real Web.

Many parameters affect the performance of the algorithms described in this paper. Ideally, we would like to perform *factorial analysis* (Montgomery, 2001) so that each combination of values is tested. Such experimentation, however, is infeasible for the large number of variables involved. We therefore take the one-factor-at-a-time approach, where we use a default value for all parameters except the one being tested. In addition, wherever appropriate, we perform several experiments testing two factors together.

6.2 Tasks And Domains

Most of our experiments are conducted in the context of several Web domains. To show the generality of our approach, we applied our methods to several additional domains, including n-queens, open knight tours, multiple robot path planning and multiple sequence alignment.

Our algorithms were applied to the following tasks:

1. *Focused crawling:* One of the main motivations for this research is the problem of *focused crawling* in the Web (Chakrabarti et al., 1999; Cho & Garcia-Molina, 2000; Kleinberg, 1999; Menczer, Pant, Srinivasan, & Ruiz, 2001). The task is to find as many goal pages as possible using limited resources, where the basic resource unit is usually the actual retrieval of a page from a link. While it looks as if the task of retrieval information from internet could have been achieved using general-purpose search engines, there are several circumstances where focused crawling is still needed:

    (a) When the search criterion is complicated and is not expressible in the query language of search engines.

    (b) When one needs an updated set of goals – search engines are updated every few weeks due to the huge space the brute-force crawlers have to cover.

    (c) When the coverage of the general engines is not sufficient.

    Previous work on focused crawling concentrated on Web-specific techniques for directing the search. Our experiments will test whether our generalization of single-goal heuristic search to multiple-goal search can contribute to the task of focused crawling.

    Performing rigorous empirical research on the Web is problematic. First, the Web is dynamic and therefore is likely to be modified between different runs of the algorithms (Douglis et al., 1997). Second, the enormous time required for crawling in the Web disallows parametric experimentation. To solve the above problems we downloaded a significant section of the Web to local storage and performed the experiments using the local copy (Cho et al., 1998; Hirai, Raghavan, Garcia-Molina, & Paepcke, 2000). Specifically, we downloaded a large part of the `.edu` domain, containing, after some cleanup, approximately 8,000,000 valid and accessible HTML pages. The resulting graph has an average branching factor of 10.6 (hyperlinks).

    We tested the performance of the algorithms on the entire downloaded domain. Some of the parametric experiments, however, were too time-consuming even when the local copy was used. Therefore, we used small sub-domains for these experiments. A sub-domain is generated by randomly selecting a root page out of a predesignated set of roots and extracting a sub-graph of size 35,000 under it.

    To ensure that the overhead of our algorithms does not significantly affect their performance, we also conducted several online experiments on the real Web.

    We use three types of goal predicates:

(a) Predicates that test for pages about specific topics: robotics, mathematics, football, food, and sport. The predicates were automatically induced by applying decision tree learning to a set of manually supplied examples.

(b) Predicates that test for certain *types* of pages: pages containing publication lists, laboratory pages, student home pages, project pages and news pages. These predicates were also learned from examples.

(c) Predicates that test for home pages of people who are members of a specific list. Each list of people was generated from a Web page listed the names of people together some personal information (such as name, affiliation and area of interest). The predicates employ commonly used heuristics for determining whether an HTML document is a home page of a specific person. We use three such predicates corresponding to three different lists we found on the Web. We limit each list to contain 100 names.

2. *Finding paths to multiple goals:* Path-finding algorithms usually search for a single path to one of the goals. In some applications, however, we get a set of initial states and a set of goal states and our task is to find a *set* of paths from an initial state to a goal state. These paths may then be used by another algorithm that evaluates them and selects one to execute according to various criteria. We simulated a physical environment by a 500×500 grid with random "walls" inserted as obstacles (an average of 210000 nodes with an average 3.9 branching factor). There are parameters controlling the maximal length of walls and the desired density of the grid. Walls are inserted randomly while making certain that the resulting graph remains connected. The set of goal states is randomly generated using one of the following two methods:

   (a) A set of states is independently and uniformly drawn from the set of all states.

   (b) 10% of the goal states are generated as above. We then randomly and uniformly select K states that are used as cluster centers. The rest of the goal states are then randomly generated with distances from the centers that are normally distributed.

   Figure 7 shows an example of a multiple path-finding problem and a solution that includes 9 paths.

3. *Planning movement of multiple robots:* Assume that we are given a set of N robots located in various states, while their task is to collect a set of $K > N$ objects scattered around. In this case we need to plan N paths that will pass through as many objects as possible. This situation is illustrated in Figure 8. Although the problem appears to resemble the one in Figure 7, the solution here has two paths (while the solution to the previous problem has 9). There are many similar planning problems: for example, that of planning product delivery from several starting points to multiple customers using a fixed number of delivery trucks. For the experiments here we used the same type of grids as for the previous problem. The robots were placed randomly. We assumed that collisions are not harmful.

4. *Open knight tours:* This is a famous problem where the task is to find a path for a knight on a chess board such that each of the squares is visited and none is visited

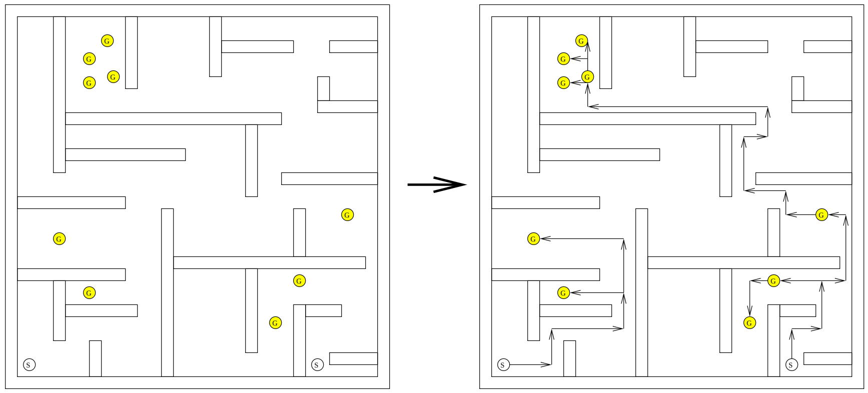


Figure 7: Searching for a set of paths to multiple goals in a grid

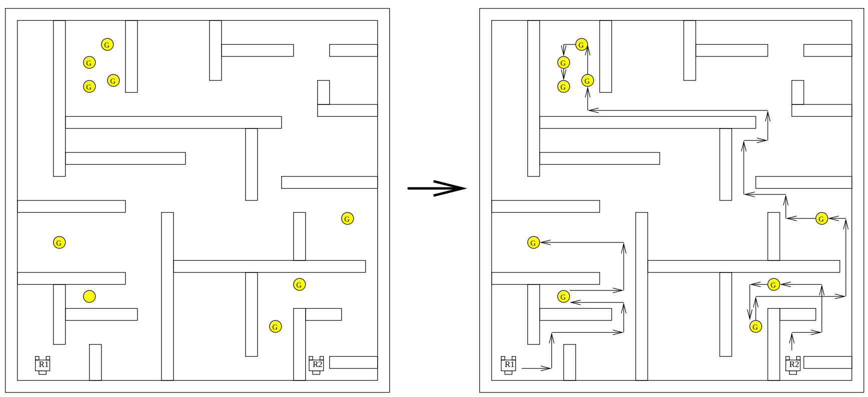


Figure 8: Multiple-robot path planning in a grid

twice. We tried a multiple-goal version of it where the task is to find as many such paths as possible within the allocated resources. For our experiments, we used boards of 6×6, 7×7 and 8×8 squares.

5. *N-Queens:* A constraint satisfaction problem where the goal is to place N queens on a chessboard such that no two queens are in the same row, column or diagonal. In the multiple goal version of this problem, we want to find as many satisfying configurations as possible within the allocated resources.

6. *Multiple sequence alignment:* A known bioinformatics problem where the goal is to align several biological sequences optimally with respect to a given cost function. In the multiple-goal version we are interested in obtaining as many almost optimal solutions as possible within the allocated resources.

6.3 Performance with Distance-Based Heuristics

We experimented first on the two multiple-goal heuristic functions that are based on graph-distance estimation: the sum heuristic and the progress heuristic. We compare the performance of multiple-goal best-first search that uses these heuristics with:

Domain	Task	BFS	Min. dist.		Sum		Progress
			Without disab.	With disab.	Without disab	With disab	
		% of goals found at 20% resources					
Multiple path finding	scattered	8.5(0.1)	21.3(1.2)	29.0(0.5)	21.5(0.9)	28.9(0.3)	76.8(2.1)
	clustered	10.2(0.1)	34.0(0.5)	45.1(0.4)	32.6(1.1)	59.2(0.3)	94(1.2)
Multiple robot movement	scattered	7.1(0.8)	20.3(0.8)	26.5(0.4)	22.3(0.6)	25.8(0.2)	89.9(1.8)
	clustered	10.1(0.9)	31.2(1.4)	47.4(1.2)	42.0(0.6)	64.1(0.7)	98.6(0.9)
		% of goals found at 2% resources					
Focused crawling 100-person search	Group 1	0.1(0.0)	13.5(0.5)	17.3(1.3)	24.1(0.7)	28.0(1.1)	51.3(0.8)
	Group 2	3.2(1.4)	18.4(2.1)	26.2(1.9)	19.7(1.0)	23.5(1.1)	78.1(3.1)
	Group 3	0.3(0.1)	5.8(0.9)	10.7(1.4)	6.4(0.9)	11.7(0.9)	60.9(0.8)

Table 1: The performance of multiple-goal search with various heuristics. The numbers in parentheses are standard deviations.

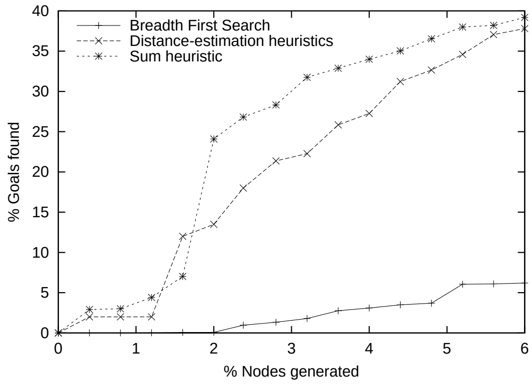
1. Multiple-goal best-first search that uses the distance estimation as its heuristic function.
2. Breadth-first search (BFS) (shown by Najork & Wiener, 2001, to perform well for Web crawling).

The sum heuristic requires distance estimates to individual goals. We define such distances for three of our domains. For the multiple path finding and multiple robot planning we use the Manhattan distance. For focused crawling task we experiment with the personal home page domain. We estimate the distance between a given page and the home page of a list member by computing the cosine vector distance between the bag of words of the given page and the bag of words of the description text for the person.

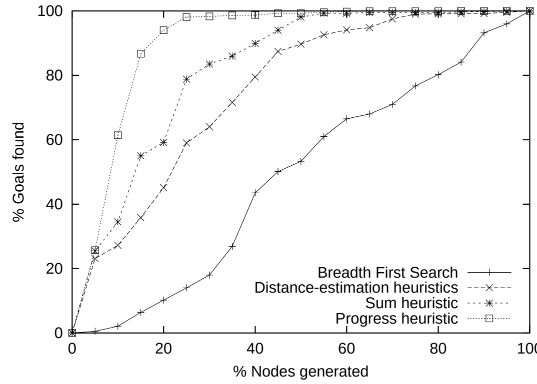
The distance heuristic and the sum heuristic were tested with and without disabling the influence of visited goals. We did not use disabling with the progress heuristic since it subsumes this behavior. For the two grid-based tasks, where all the goals are given, we used complete disabling by removing the visited goals from the goal list as described in Section 4.5.

For the personal home page search task, where the set of goals is not explicitly given, we used the feature-based method described in Section 4.5. The features used for determining the distance between candidate pages and visited goals are the words with the highest *TFIDF* value (Joachims, 1997; Salton & Buckley, 1988).

Table 1 summarizes the results of this experiment. Each number represents the average of 50 experiments. We measure the performance after consuming 20% of the maximal amount of resources, i.e., expanding 20% of the total number of nodes in the search graph. The 100-person home page search proved to be a relatively easy domain where our methods were able to find most of the goals after consuming very little resources. Therefore, for this domain, we measure the performance at 2% of the nodes. Figure 9 shows the anytime behavior of the various methods for two of the domains. The graphs for the other domains show similar patterns.



(a)



(b)

Figure 9: Anytime performance of the various heuristics: (a)Focused crawling (b)Multiple path finding

On the basis of this table and corresponding graphs, we make the following observations:

1. The progress heuristic is superior to all other methods tested so far. That it is advantageous for the case of clustered goals comes as no surprise because it leads to one cluster being pursued at a time. That it is superior to distance estimation for the case of scattered goals is far less obvious: we would expect both heuristics to pursue one goal after another and therefore yield similar results. The weighted progress heuristic, however, prefers pursuing goals that are closer to other goals, thus yielding better results.

2. The results clearly demonstrate that the goal influence phenomenon is indeed significant, and our method is efficient in reducing this effect.

3. In almost every case, heuristic methods are significantly better than blind search. The only exception is when using the sum heuristic without influence disabling in graphs with scattered goals. In such cases, its behavior is only marginally better than blind search.

6.4 Performance with Marginal-Utility Heuristics

None of the methods in the previous subsection take into account the expected search resources involved in pursuing the alternative directions. In addition, they all assume either knowledge of the specific set of goals or of the heuristic distances to each. In this subsection we test the performance of the two methods that are based on marginal utility as described in Section 5.1.2. The experiments described in this subsection are performed for the focused crawling task with the 10 goal predicates described in Section 6.2.

For the topic-based goals, we cannot compare the performance of the marginal-utility algorithms to that of the sum heuristic because we do not have access to the list of goals or to a list of heuristic values for specific goals as in the previously tested domains. Therefore we use for comparison blind BFS and the common best-first search using the distance-estimation heuristic. The heuristic is based on the list of words selected by the induction algorithm when generating the goal predicates.

6.4.1 Inferring Marginal Utility from Partial Marginal Utility

We implemented and tested the marginal-utility estimation algorithm described in Section 5.1. Figure 10 shows the performance profiles of the tested algorithms for the *robotics* and

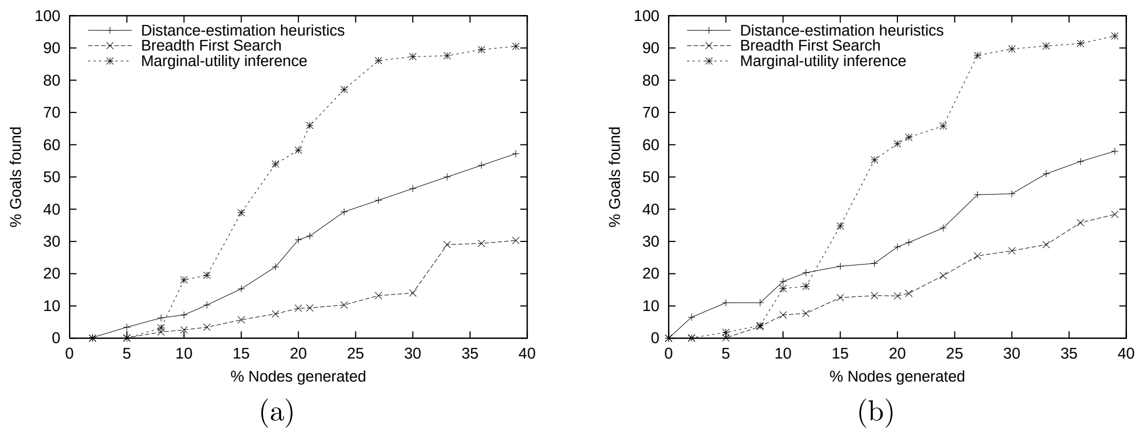


Figure 10: The performance of multiple-goal best-first search using the marginal-utility inference method applied to focused crawling (with D=4). The results are shown for (a)Robotics pages (b) Mathematics pages

mathematics goal predicates. Each graph represents the average of 5 runs of the tested algorithm using 5 starting pages randomly selected from the fixed 200 root pages. In both cases we see a significant advantage of the marginal-utility method over the other two methods. This advantage becomes evident only after an initial "training" period, in which sufficient statistics are accumulated. The full data for the 10 domains with resource allocation of 10% and 20% is available in the Appendix. Figure 11 shows the average improvement factor (compared to distance estimation) for the 10 domains as a function of the search resources. The graph shows very nicely the initial "exploratory" stage where the statistics are accumulated until about 8% of the resources have been consumed, after which the improvement factor becomes larger than 1. The improvement factor reaches a peak of about 2.8 at 17%, and then starts to decline towards a value of 1 at 100% search resources, where any algorithm necessarily finds all the goals.

Performance at the exploratory stage can be improved by combining the two methods. We tested a hybrid method, which uses a linear combination of the marginal-utility prediction and the heuristic estimation. To determine the linear coefficients of each part, we conducted 100 experiments on the small Web subgraph (below 35,000 pages) using different goal predicates (unrelated to those tested in our main experiments). Figure 12 shows the results obtained for the combined method compared to each of the individual methods. We can see that indeed the combined method is better than each of the algorithms alone. An

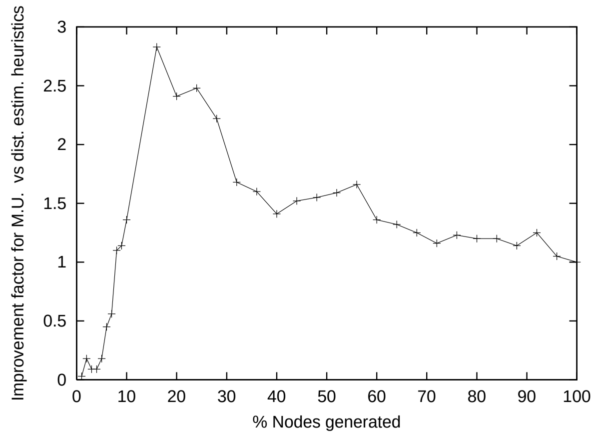


Figure 11: The improvement factor for best-first using marginal-utility inference compared to best-first using the distance-estimation heuristic

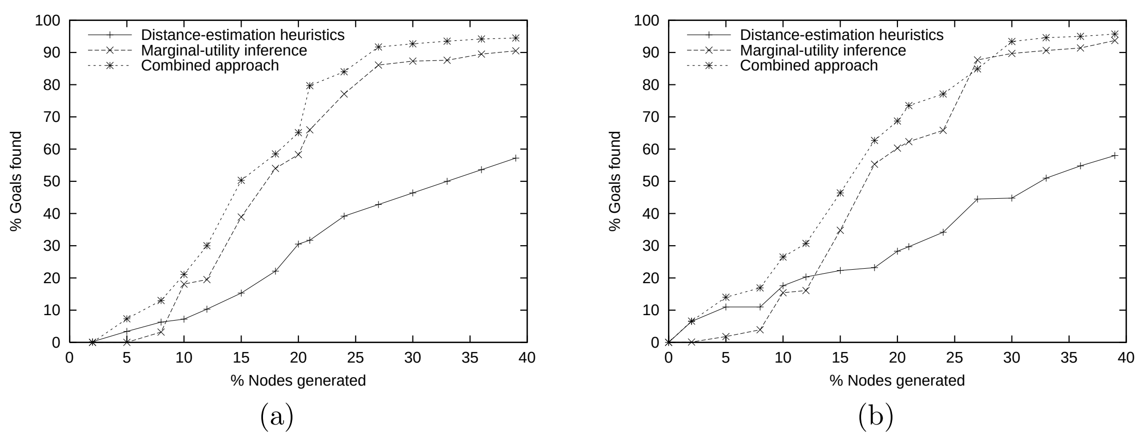


Figure 12: Combining marginal-utility inference with heuristic search for (a)Robotics pages (b)Mathematics pages.

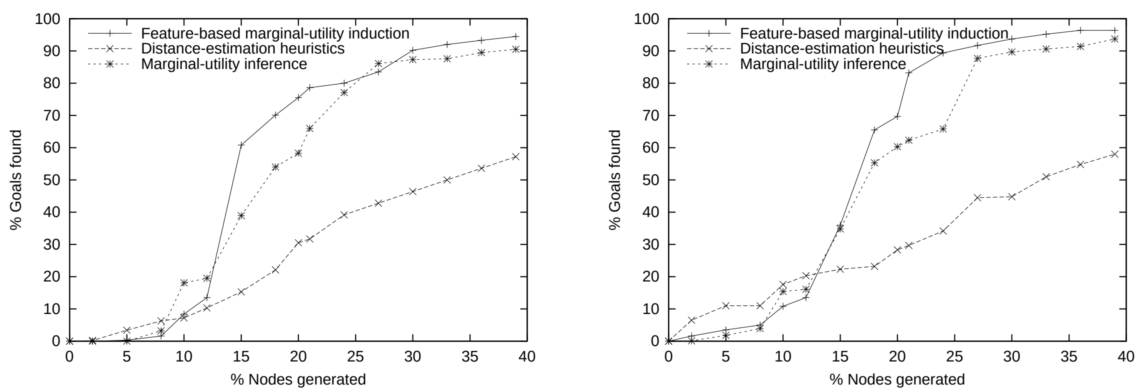
interesting phenomenon is that, at all points, the result for the combined method is better than the maximal results for the other two. One possible explanation is that very early in the search process the distance-estimation heuristic leads to a sufficient number of goals to jump-start the learning process much earlier. The results for the 10 domains are available in the Appendix.

6.4.2 Learning Marginal-Utility from Features

In Section 5 we describe a method for feature-based generalization of visited search nodes in order to induce marginal-utility values. We conducted a set of experiments to test the efficiency of the learning mechanism for the problem of focused crawling with the same 10 goal types as in previous experiments.

During crawling, we accumulate the *supported* visited pages and tag them with their marginal utility as measured at the time learning took place (see Section 5). We then convert the tagged pages to feature vectors and hand them to the CART algorithm (Breiman et al., 1984) for regression-tree induction[6]. We then used the induced tree to estimate the marginal utility of newly generated nodes.

For features, we use the bag-of-words approach, which is dominant in the field of text categorization and classification. The value of each word-feature is its appearance frequency. Words appearing in HTML title tags are given more weight. We apply feature selection to choose words with the highest *TFIDF* (Joachims, 1997; Salton & Buckley, 1988).



(a) (b)

Figure 13: The performance of best-first search with marginal utility induced using the regression trees classifier. The experiments were performed on the problem of focused crawling with: (a)Robotics pages (b) Mathematics pages

Figure 13 shows the results obtained for the *robotics* and *mathematics* goal predicates. The full report for the 10 domains is available in the Appendix. In both cases, there is an initial period where the sibling-based inference method outperforms the more sophisticated feature-based induction. After this period, however, the induction-based method significantly outperforms the sibling-based method. One possible reason for the initial inferior performance is that feature-based induction requires more examples than the simplistic sibling-based method that only computes averages and therefore needs fewer examples.

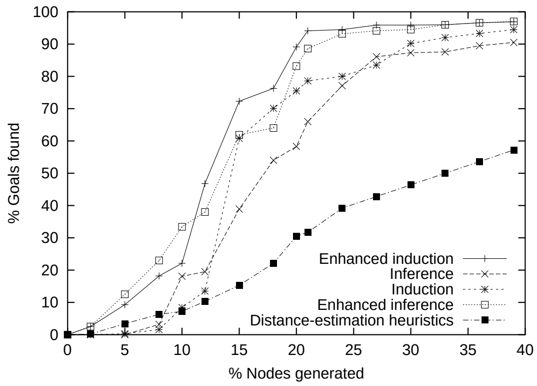
We inspected the produced trees and found out that they reflect reasonable concepts. They have several dozens of nodes and contain both features that are related to the searched goal and features that correspond to hubs (such as *repository* and *collection*).

We have tested whether our choice of classifier affects the performance of the induction-based method by performing the same set of experiments using the KNN classifier. The results obtained were essentially identical.

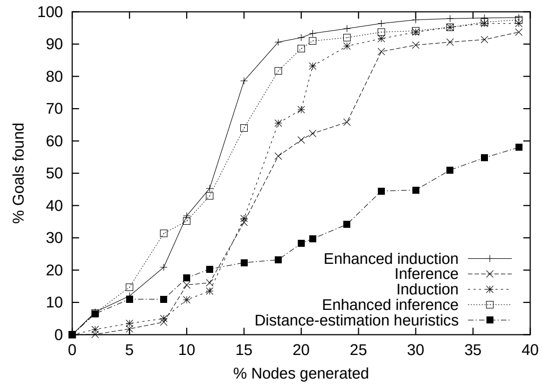
6. Other induction algorithms, such as SVM or Naive Bayes, could have been used as well.

6.5 Testing the Full System Architecture

We have described various enhancements of our marginal-utility methods, including sibling clustering, overlap minimization, combining the marginal-utility heuristic with the distance-estimation heuristic, and disabling of visited goals. Figure 14 shows the performance of the two marginal-utility methods with the full set of enhancements. The full results for the 10 domains are available in the Appendix.



(a)



(b)

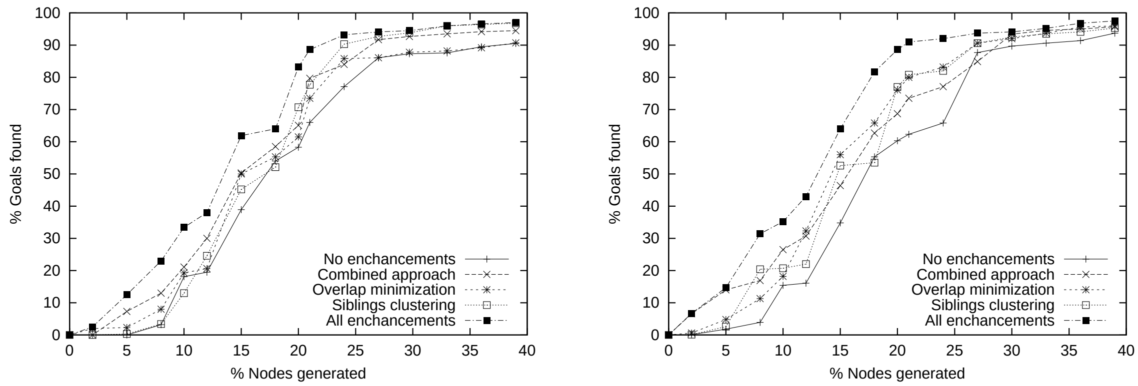
Figure 14: The performance of best-first search (using different marginal-utility heuristics) with all the enhancements enabled: (a)Robotics pages (b) Mathematics pages. Each figure contains 5 plots: one for the baseline performance (*distance estimation*), two for the unenhanced methods (*inference* and *induction*) and two for the enhanced methods (*enhanced inference* and *enhanced induction*).

Both the sibling- and feature-based methods indeed improve when all the enhancements are enabled. Furthermore, the feature-based method maintains its advantage, albeit with a slightly decreased magnitude. Although enabling all the enhancements does improve system performance, it should be recalled that the same is true for enabling each enhancement separately. A question thus arises whether the improvements are at least partially cumulative. In other words, would performance using *all* the enhancements be better than performance using each of the enhancements separately? Figure 15 compares all the graphs for the sibling-based method. We can see that the fully enhanced method is indeed superior to all the rest.

6.6 Realtime Performance

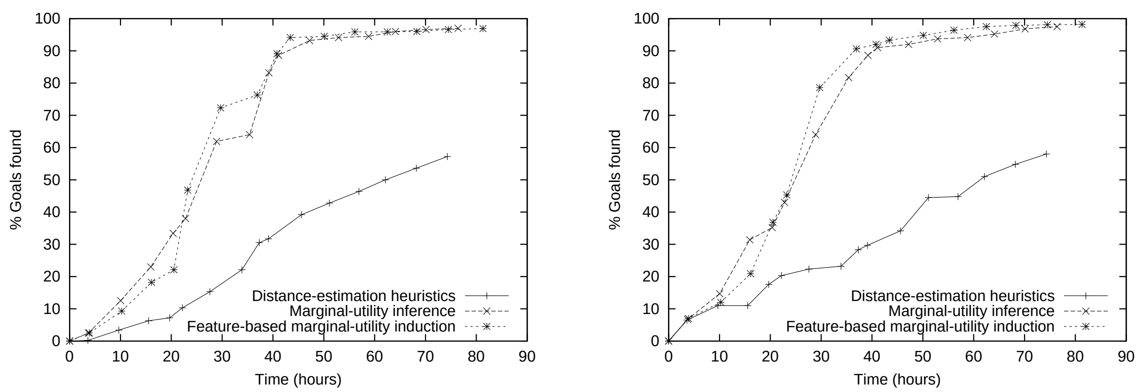
In the previous experiments we took the number of generated nodes as a basic resource unit. We must be careful, however, since this measurement does not take into account the overhead in our method. To ensure that the overhead does not outweigh the benefits, we conducted a realtime evaluation of the system architecture performing focused crawling on the online Web and measured the resources consumed in the time that elapsed.[7]. Figures

7. The experiment was performed using a Pentium-4 2.53 GHz computer with 1GB of main memory and a cable modem.



(a) (b)

Figure 15: The performance of best-first search (using marginal-utility inference) with all the enhancements enabled compared to the performance of the algorithm with a single option enabled: (a)Robotics pages (b) Mathematics pages. Each figure contains 5 plots: One for the marginal utility inference method with no enhancements, one for the same method enhanced by the combined approach, one for enhancement by overlap estimation, one for enhancement by sibling clustering and, finally, one for all the enhancements together.



(a) (b)

Figure 16: The performance of the marginal-utility based methods as a function of real time: (a)Robotics pages (b)Mathematics pages

16(a),(b) show the performance of our methods with all the enhancements enabled as a function of real time. A comparison of these graphs to the graphs shown in Figure 14 reveals that the overhead of our methods does not noticeably affect their performance. To see if the overhead increases with the size of the visited graph, we plotted in Figure 17 the real time as a function of the number of generated nodes. The graphs show that the majority of the time required for focused crawling tasks is indeed the loading time itself,

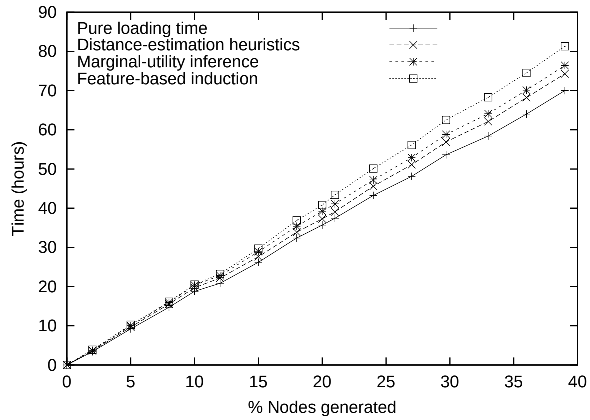


Figure 17: The average real time used by the marginal-utility methods as a function of the number of generated nodes in the focused crawling domain

even when all calculations related to the discussed options are enabled. In fact, using distance-estimation increases the computation time by a factor of no more than 1.07; using similarity-based inference of marginal utility, by no more than 1.1; and using feature-based induction, by no more than 1.17. Thus, if our improvement is greater than these factors, our algorithms will benefit focused crawling systems.

6.7 Contract Algorithms Versus Interruptible Anytime Algorithms

The experiments described so far test the performance of our methods as interruptible anytime algorithms. Each point of the graph can also be considered as a test result for a contract algorithm using the specific resource allocation. However, if a multiple-goal search algorithm is called in contract mode, we can utilize the additional input (of resource allocation) to improve the performance of our algorithm, as described in Section 5.1.2.

To test the effect of exploiting the resource allocation, we repeated the experiment described in Section 6.4.1 using the algorithm of Section 5.1.2. We found out that when using the algorithm in contract mode, we obtained an average improvement factor of 7.6 (factor of 2.8 if dropping two extremes) for 5% resource allocation and 1.4 for 10% resource allocation. The full results are available in the Appendix.

Our algorithms also allow a "contract by quality" mode where the input is the required quality instead of the allocated resources. In our case the quality is specified by the percentage of goals found. The contract algorithm achieved an average improvement factor of 1.9 for 5% of the goals and 1.4 for 20% of the goals. The full results are available in the Appendix.

6.8 The Performance of Other Multiple-Goal Search Algorithms

In previous subsections we test our heuristic methods on best-first search. To show the generality of our approach, we test whether marginal-utility heuristics can be used efficiently for dynamic ordering of variables in backtracking, and for choosing a node from the focal group in the multiple-goal A^*_ϵ algorithm. In both cases we used the sibling clustering method.

For backtracking we used the multiple-goal version of two known problems: open knight tour and n-queens. We applied our marginal-utility inference algorithm which based on similarity of siblings to sort variable values in the multiple-goal version of backtracking search. For the open knight tour problem we used a 6×6 board (this board contains $524,486$ goal configurations, and this is the maximal size where we can collect all the goals efficiently). For the n-queens problem we used a 16×16 board containing $14,772,512$ goals. In neither case did we use a domain-specific technique to increase search efficiency. Figure

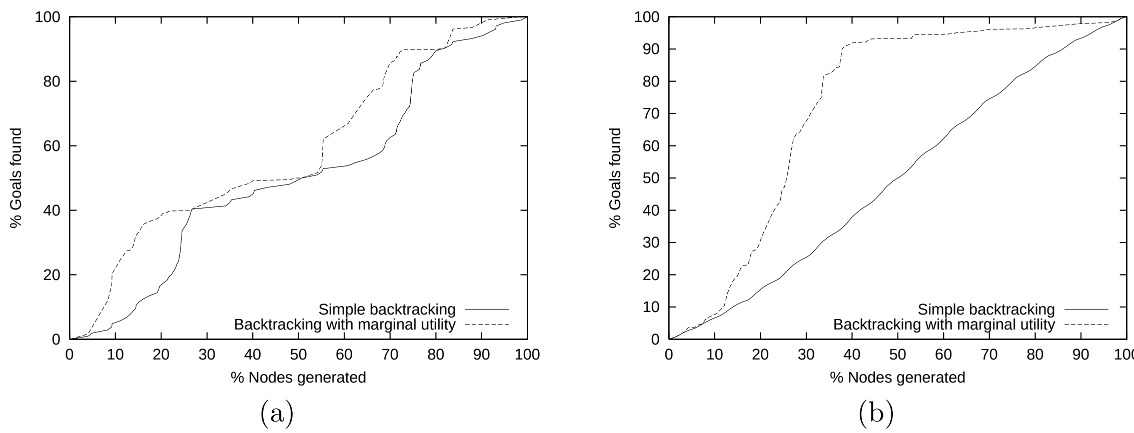


Figure 18: Applying marginal-utility inference to backtracking search (a)Open knight tour (b) N-queens task

18(a),(b) compares the performance of multiple-goal backtracking search with and without marginal utility. We can see that applying marginal-utility inference significantly increases the performance for both problems.

We also tested the multiple-goal A^*_ϵ algorithm, described in Section 3, where the sibling-based marginal-utility heuristic selects a node from the focal group. The experiment was performed on the known multiple-sequence alignment problem. Since the graph degree is very large, we applied a modified version of A^* described by Yoshizumi, Miura and Ishida (2000). We used the same heuristic, methodology and data set described in this paper, requiring the algorithm to collect all optimal and 1.1-suboptimal solutions.

Figure 19 shows the results obtained for two data sets. We can see that marginal utility improves the performance of the search algorithm.

7. Related Work

The multiple-goal search framework defined in this paper is novel. No previous work has treated heuristic search for multiple goals as a general search framework and no previous work provided *general* algorithms for multiple-goal search. The planning community has dealt with multiple goals only in an entirely different setup, where the goals are conjunctive

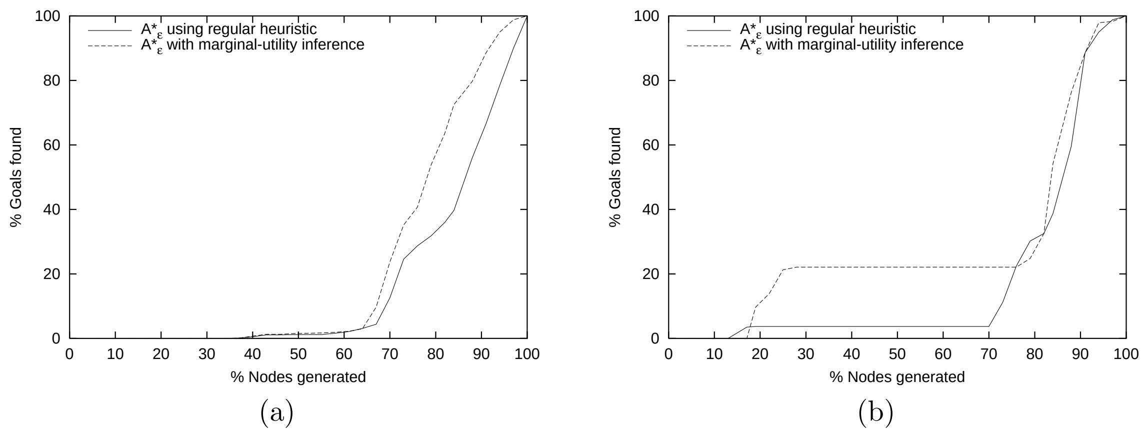


Figure 19: Applying marginal-utility inference to A^*_ϵ search (a)Data set 1 (b) Data set 2

and possibly conflicting. In particular, that setup is not easily applicable to generic graph search.

Even domain-specific algorithms for multiple-goal heuristic search are not very common. Of the domains mentioned in Section 6.2, the only one that given any attention as a multiple-goal problem domain was Web crawling. The sequence alignment domain was used in several works on heuristic search (for example, Korf & Felner, 2002; Zhou & Hansen, 2002, 2003; Schroedl, 2005), but only as a single-goal search problem.

The popularity of the Web has led many researchers to explore the problem of multiple-goal search in the Web graph. This problem is better known as focused crawling. Chakrabarti et al. (1999) defined a focused crawler as "a Web agent which selectively seeks out pages that are relevant to a pre-defined set of topics by retrieving links from the live Web." Such agents can be used for building domain-specific Web indices (see for example McCallum, Nigam, Rennie, & Seymore, 1999). Focused Web-crawling algorithms use various methods such as breadth-first search (Najork & Wiener, 2001), best-first search (Cho & Garcia-Molina, 2000; Cho et al., 1998), and reinforcement learning (Boyan, Freitag, & Joachims, 1996; Rennie & McCallum, 1999). Most of the heuristic methods for focused crawling are based on Web-specific features. For example, the page-rank model (Page, Brin, Motwani, & Winograd, 1998) was used by Brin and Page (1998), and by Haveliwala (1999). The hubs-and-authorities model (Kleinberg, 1999) was used by Borodin, Roberts, Rosenthal, and Tsaparas (2001). In addition, several theoretical works provide analysis and bounds for the problem of Web crawling (for example Cooper & Frieze, 2002; Kumar et al., 2000).

The approach most similar to ours is that taken by by Rennie and McCallum (Rennie & McCallum, 1999), who apply reinforcement learning to Web crawling. Like our marginal-utility induction method, their method also estimates a reward value and generalizes it over unvisited nodes. There are, however, several important differences between the two methods, particulary in the definition of reward. While our approach is based on the maximal number of goals achieved for the given resources, their method is focused on immediacy of goal achievement. The sooner a goal can be achieved by an optimal algorithm starting from a graph node, the more it contributes to the reward value of this node regardless of whether the algorithm has enough resources to collect this goal. Thus, their setup does not allow a direct incorporation of supplied resource limit input. Furthermore, their approach relies on relatively heavy off-line processing on a training set. We propose an online update method to estimate and update the marginal-utility based system. Our approach not only eliminates the need for fetching the fixed training set, but also gives more flexibility to the algorithm.

8. Discussion

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 show that the traditional distance-estimation heuristic is not sufficiently effective for multiple-goal search. We then introduce the *sum* and *progress* heuristics, which take advantage of an explicitly given goal set to estimate the direction to larger and closer groups of goals.

One problem with the above heuristics is that they ignore the expected resources required to collect goals in the alternative directions. We introduce the marginal-utility heuristic, which attempts to estimate the cost per goal of each search direction. Thus, using it should lead to a more productive search.

Designing an effective marginal-utility heuristic is a rather difficult task. We therefore developed two methods for online learning of marginal-utility heuristics. One is based on local similarity of the partial marginal-utility of sibling nodes, and the other generalizes marginal-utility over the state feature space. Both methods infer marginal utility from the partial marginal-utility values which are based on the number of visited goal and non-goal nodes in partially explored subgraphs.

The sibling-based inference method requires only the basic input of a search problem: a set of starting nodes, a successor function, and a goal predicate. The method can also take advantage of an input resource allocation, as was demonstrated in Section 6.7. If a distance-estimation heuristic is given, the sibling-based method can utilize it for the initial stages of the search where the data on which to base the inference is not sufficient. The marginal-utility generalization method requires a set of meaningful features over the set of states. This is a common requirement for learning systems.

We applied our methodology to several tasks, including focused Web crawling, and showed its merit under various conditions. We also applied it to the tasks of finding paths to multiple goals, planning movement of multiple robots, knight-tour, n-queens, and finding a set of multiple sequence alignments. The experiments show that even without any prior knowledge about the goal type, and given only the goal predicate, our algorithm, after an initiation period, significantly outperforms both blind and best-first search using a distance-estimation heuristic. When we enhance our method with a regular distance-estimation heuristic, our method shows more than threefold improvement over the same distance-estimation heuristic alone.

Our framework and proposed algorithms are applicable to a wide variety of problems where we are interested in finding many goal states rather than only one. We show, for example, for the multiple sequence alignment problem, that our methods allow A^*_ϵ to reach many more nearly-optimal configurations. The same methods can be also applied to constraint satisfaction problems where it may be useful to find many solutions and apply another algorithm for selecting a solution from the found set.

To apply our framework to new problems, the following requirements must be fulfilled:

1. The problem domain should be formulated as a state space.

2. There exists predicate that identifies goal states.

3. For using the sum and progress heuristics:

   (a) A traditional function that estimates the distance between two states should be given.

   (b) The set of goals states should be given explicitly.

4. For the sibling-based method of marginal utility inference we assume that the marginal utility values of sibling nodes are relatively similar.

5. For the induction-based marginal utility inference we assume the availability of a set of state features that are informative with respect to marginal utility.

The main message of this research is that the induction-based method for marginal utility inference should be used when possible. Unlike the sum and the progress heuristics, it takes into account the resources needed for collecting the goals. Unlike the sibling-based inference method, it makes no assumptions about similarity of marginal utility values between siblings. It does require informative state features; however, for many domains, a reach set of state features – sufficient for inducing the marginal utility estimation function – is available.

Our marginal-utility based heuristic techniques are greedy in the sense that they always choose the node leading to a subgraph where we would expect to find the maximal number of goals, were we to use all the remaining resources for that subgraph. A more sophisticated approach would try to wisely distribute the remaining resources between promising directions, leading, we hope, to better performance than the greedy approach.

Although the marginal utility approach does not consider possible overlap between subgraphs, we propose, in Section 4.5, a technique to reduce it. A more interesting direction could be to predict the actual overlap as the search progresses, using the same methods as in the partial marginal-utility calculation.

The framework described in this paper opens a new research direction. The field is wide open for the development of new algorithms and for the application of multiple-goal search algorithms to other tasks.

Acknowledgements

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.

Appendix A. Detailed Results

In this appendix we provide a breakdown of the results for the 10 Web topics. Tables 2 and 3 refer to the results described in Section 6.4.1. Table 4 refers to the results described in Section 6.4.2. Table 5 refers to the results described in Section 6.5. Tables 6 and 7 refers to the results described in Section 6.7.

Goal type	% of goals found at 10% resources			% of goals found at 20% resources		
	BFS	Distance estimation	Marginal utility	BFS	Distance estimation	Marginal utility
Robotics	2.6(0.9)	7.2(0.2)	18.1(0.6)	9.3(1.6)	30.5(1.1)	58.3(2.0)
Students	10.5(1.0)	11.8(0.5)	17.0(1.3)	12.5(1.5)	23.7(0.6)	54.6(1.3)
Mathematics	7.2(0.1)	17.6(1.6)	15.4(0.8)	13.1(1.2)	28.3(0.8)	60.3(2.9)
Football	0.0(0.0)	31.4(0.5)	25.3(0.9)	0.8(0.0)	42.1(1.7)	71.5(0.7)
Sports	3.6(0.4)	37.0(1.1)	45.1(2.7)	16.2(0.7)	41.6(0.7)	68.5(1.4)
Laboratories	0.3(0.1)	12.5(0.7)	33.4(0.8)	5.3(0.5)	18.1(0.8)	80.4(3.1)
Food	7.2(1.2)	25.1(0.9)	30.3(1.7)	26.5(3.9)	48.5(1.7)	92.7(2.2)
Publications	0.3(0.1)	12.6(1.0)	35.2(2.6)	3.5(0.1)	18.5(1.1)	64.2(1.9)
Projects	0.1(0.0)	30.1(1.4)	41.2(1.6)	0.8(0.2)	32.0(1.4)	80.5(2.0)
News	8.5(0.9)	23.1(0.8)	22.6(0.8)	15.3(1.0)	40.4(1.2)	88.9(0.7)

Table 2: Marginal-utility and distance-estimation heuristics in focused crawling (with D=4). The numbers in parentheses are standard deviations.

Goal type	% of goals found at 10% resources			% of goals found at 20% resources		
	Distance estimation	Marginal utility	Combined method	Distance estimation	Marginal utility	Combined method
Robotics	7.2(0.2)	18.1(0.6)	21.1(0.5)	30.5(1.1)	58.3(2.0)	65.1(2.1)
Students	11.8(0.5)	17.0(1.8)	23.3(0.6)	23.7(0.6)	54.6(1.3)	59.6(1.2)
Mathematics	17.6(1.6)	15.4(0.8)	26.5(1.9)	28.3(0.8)	60.3(2.9)	68.7(2.5)
Football	31.4(0.5)	25.3(0.9)	33.9(1.0)	42.1(1.7)	71.5(0.7)	70.9(1.1)
Sports	37.0(1.1)	45.1(2.7)	48.3(2.2)	41.6(0.7)	68.5(1.4)	75.0(1.3)
Laboratories	12.5(0.7)	33.4(0.7)	40.2(0.9)	18.1(0.8)	80.4(3.1)	79.8(1.0)
Food	25.1(0.9)	30.3(1.7)	30.1(1.2)	48.5(1.7)	92.7(2.2)	93.3(1.6)
Publications	12.6(1.0)	35.2(2.7)	42.0(2.4)	18.5(1.1)	64.2(1.9)	77.8(1.7)
Projects	30.1(1.4)	41.2(1.6)	41.3(1.0)	32.0(1.4)	80.5(2.0)	84.7(1.8)
News	23.1(0.8)	22.6(0.8)	30.5(1.9)	40.4(1.2)	88.9(0.7)	90.5(0.9)

Table 3: Combining marginal-utility with distance-estimation heuristics for focused crawling. The numbers in parentheses are standard deviations.

Goal type	% of goals found at 10% resources			% of goals found at 20% resources		
	Distance estim.	Inference by siblings	Inference by induction	Distance estim.	Inference by siblings	Inference by induction
Robotics	7.2(0.2)	18.1(0.7)	8.4(1.2)	30.5(1.1)	58.3(2.0)	75.5(2.5)
Students	11.8(0.6)	17.0(1.9)	12.4(1.1)	23.7(0.6)	54.6(1.3)	68.2(1.6)
Mathematics	17.6(1.6)	15.4(0.9)	10.8(1.5)	28.3(0.8)	60.3(2.9)	69.7(2.5)
Football	31.4(0.5)	25.3(0.4)	21.1(0.7)	42.1(1.7)	71.5(0.7)	71.6(1.1)
Sports	37.0(1.2)	45.1(2.7)	36.2(1.6)	41.6(0.7)	68.5(1.4)	79.4(1.5)
Laboratories	12.5(0.7)	33.4(0.8)	19.4(1.3)	18.1(0.8)	80.4(3.1)	86.0(1.3)
Food	25.1(0.9)	30.3(1.7)	27.1(0.6)	48.5(1.7)	92.7(2.2)	89.1(1.8)
Publications	12.6(1.0)	35.2(2.7)	21.8(2.4)	18.5(1.1)	64.2(1.9)	78.9(1.2)
Projects	30.1(1.4)	41.2(1.8)	35.5(1.2)	32.0(1.4)	80.5(2.0)	89.8(2.5)
News	23.1(0.8)	22.6(0.8)	23.2(0.9)	40.4(1.2)	88.9(0.7)	93.9(1.1)

Table 4: The performance of the two methods of marginal-utility estimation for the focused crawling task. The numbers in parentheses are standard deviations.

Goal type	Distance-estimation	Inference by siblings		Inference by induction	
		Without enhanc.	With enhanc.	Without enhanc.	With enhanc.
	% of goals found at 10% resources				
Robotics	7.2(0.2)	18.1(0.7)	33.4(2.1)	8.4(1.2)	22.1(2.2)
Students	11.8(0.5)	17.0(1.8)	26.5(1.0)	12.4(1.1)	25.2(1.6)
Mathematics	17.6(1.6)	15.4(0.8)	35.2(1.4)	10.8(1.4)	36.8(1.5)
Football	31.4(0.6)	25.3(0.9)	46.9(1.4)	21.1(0.7)	39.5(1.4)
Sports	37.0(1.2)	45.1(2.7)	54.9(1.6)	36.2(1.6)	46.8(1.5)
Laboratories	12.5(0.7)	33.4(0.8)	46.4(1.6)	19.4(1.3)	39.5(1.4)
Food	25.1(0.9)	30.3(1.7)	35.1(1.4)	27.1(0.6)	30.4(1.4)
Publications	12.6(1.1)	35.2(2.6)	46.8(1.5)	21.8(2.4)	30.0(1.4)
Projects	30.1(1.4)	41.2(1.6)	49.2(1.2)	35.5(1.2)	44.6(1.3)
News	23.1(0.8)	22.6(0.8)	41.1(1.6)	23.2(0.9)	42.3(1.1)
	% of goals found at 20% resources				
Robotics	30.5(1.2)	58.3(2.0)	83.2(1.8)	75.5(2.6)	89.1(1.5)
Students	23.7(0.6)	54.6(1.3)	65.4(1.7)	68.2(1.6)	78.3(1.6)
Mathematics	28.3(0.8)	60.3(2.9)	88.6(1.0)	69.7(2.5)	92.0(1.1)
Football	42.1(1.7)	71.5(0.8)	80.6(1.4)	71.6(1.1)	91.3(1.4)
Sports	41.6(0.7)	68.5(1.4)	78.5(1.6)	79.4(1.6)	86.4(1.5)
Laboratories	18.1(0.8)	80.4(3.1)	86.4(2.0)	86.0(1.3)	92.1(1.4)
Food	48.5(1.7)	92.7(2.2)	95.1(1.6)	89.1(1.8)	95.0(1.7)
Publications	18.5(1.1)	64.2(1.9)	87.2(2.1)	78.9(1.2)	94.7(1.4)
Projects	32.0(1.4)	80.5(2.1)	85.4(1.7)	89.8(2.5)	95.2(1.7)
News	40.4(1.3)	88.9(0.8)	93.8(1.2)	93.9(1.1)	96.5(1.5)

Table 5: The performance of the two marginal-utility based methods with all the enhancements enabled. The numbers in parentheses are standard deviations.

Goal type	% of goals found					
	Contract: 5% Resources		Contract: 10% Resources		Contract: 20% Resources	
	Anytime	Contract	Anytime	Contract	Anytime	Contract
Robotics	0.1(0.0)	2.9(0.2)	18.1(0.7)	24.6(1.1)	58.3(2.0)	69.5(2.2)
Students	0.7(0.0)	1.9(0.1)	17.0(1.9)	29.3(1.5)	54.6(1.3)	62.8(1.3)
Mathematics	1.8(0.3)	8.4(0.8)	15.4(0.8)	24.1(0.9)	55.3(0.9)	64.0(1.1)
Football	4.2(0.2)	12.1(0.5)	25.3(0.9)	29.9(1.0)	71.5(0.8)	75.6(0.8)
Sports	12.7(0.6)	20.4(1.1)	45.1(2.7)	60.4(1.1)	68.5(1.4)	77.7(1.3)
Laboratories	10.5(1.0)	11.3(0.6)	33.4(0.7)	40.7(0.9)	80.4(3.1)	81.6(0.1)
Food	9.4(0.9)	22.1(1.2)	30.3(1.7)	32.1(1.6)	92.7(2.2)	92.8(2.2)
Publications	2.1(0.3)	12.9(1.7)	35.2(2.6)	55.0(2.3)	64.2(1.9)	76.2(1.5)
Projects	0.9(0.0)	21.5(1.6)	41.2(1.6)	48.6(0.9)	80.5(2.1)	86.5(1.6)
News	8.8(1.0)	13.2(0.9)	22.6(0.8)	41.5(1.0)	88.9(0.7)	91.1(1.0)

Table 6: Contract vs. anytime performance when supplying a resource limit. The numbers in parentheses are standard deviations.

Goal type	% of resources consumed					
	Contract: 5% Goals		Contract: 20% Goals		Contract: 50% Goals	
	Anytime	Contract	Anytime	Contract	Anytime	Contract
Robotics	8.4(0.6)	6.2(1.1)	12.1(0.7)	9.8(0.5)	17.2(1.2)	10.2(1.0)
Students	9.0(0.5)	7.1(0.4)	11.3(0.5)	8.4(0.3)	18.7(0.6)	9.9(0.5)
Mathematics	5.4(0.4)	3.9(0.2)	10.3(0.5)	6.8(0.4)	14.3(0.6)	9.9(0.4)
Football	5.2(0.2)	2.1(0.2)	8.4(0.3)	5.2(0.2)	14.6(0.5)	9.9(0.7)
Sports	3.9(0.2)	1.9(0.1)	7.1(0.6)	6.0(0.4)	12.2(0.6)	6.5(0.6)
Laboratories	4.2(0.1)	1.8(0.1)	7.4(0.4)	5.3(0.3)	14.1(0.6)	6.6(0.2)
Food	5.9(0.3)	2.2(0.3)	8.8(0.5)	5.1(0.2)	14.3(0.5)	7.0(0.6)
Publications	6.1(0.5)	3.3(0.4)	7.7(1.1)	5.9(0.5)	18.8(0.7)	10.3(0.6)
Projects	4.0(0.3)	1.7(0.1)	8.3(0.5)	6.1(0.3)	16.5(0.6)	9.0(0.5)
News	7.6(0.7)	5.9(0.4)	9.8(0.2)	6.5(0.3)	16.0(0.4)	10.3(0.6)

Table 7: Contract vs. anytime performance when supplying goal requirements. The numbers in parentheses are standard deviations.

References

Boddy, M., & Dean, T. L. (1994). Deliberation scheduling for problem solving in time constrained environments. *Artificial Intelligence*, *67*(2), 245–285.

Borodin, A., Roberts, G. O., Rosenthal, J. S., & Tsaparas, P. (2001). Finding authorities and hubs from link structures on the www. In *The 10th International WWW conference*, pp. 415–429. ACM Press.

Boyan, J., Freitag, D., & Joachims, T. (1996). A machine learning architecture for optimizing web search engines. In *Proceedings of the AAAI Workshop on Internet-Based Information Systems*, pp. 324–335.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, P. J. (1984). *Classification and Regression Trees*. Wadsworth International, Monterey, CA.

Brin, S., & 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., & Dom, B. (1999). Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks*, *31*(11–16), 1623–1640.

Cho, J., & 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., & Whang, K.-Y. (Eds.), *VLDB 2000*, pp. 200–209, Los Altos, CA 94022, USA. Morgan Kaufmann Publishers.

Cho, J., García-Molina, H., & Page, L. (1998). Efficient crawling through URL ordering. *Computer Networks and ISDN Systems*, *30*(1–7), 161–172.

Cooper, C., & Frieze, A. (2002). Crawling on web graphs. In *Proceedings of STOC*, pp. 419–427.

Cover, T. M., & Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*, 21–27.

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, *14*(4), 318–334.

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., & Gori, M. (2000). Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, pp. 527–534, Cairo, Egypt.

Douglis, F., Feldmann, A., Krishnamurthy, B., & Mogul, J. C. (1997). Rate of change and other metrics: a live study of the www. In *USENIX Symposium on Internet Technologies and Systems*, pp. 147–158.

Gasser, R. U. (1995). *Harnessing Computational Resources for Efficient Exhaustive Search*. Ph.D. thesis, ETH, Swiss Federal Institute of Technology, Zurich, Switzerland.

Hansen, E. A., & Zilberstein, S. (1996). Monitoring the progress of anytime problem-solving. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pp. 1229–1234, Portland, Oregon, USA. AAAI Press / The MIT Press.

Haveliwala, T. (1999). Efficient computation of PageRank. Tech. rep. 1999-31.

Held, M., & Karp, R. M. (1970). The traveling salesman problem and minimum spanning trees. *Operations Research*, *18*, 1138–1162.

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

Hovitz, E. (1990). *Computation and Action under Bounded Resources*. Ph.D. thesis, Stanford University.

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

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

Korf, R. E., & Zhang, W. (2000). Divide-and-conquer frontier search applied to optimal sequence allignment. In *National Conference on Artificial Intelligence (AAAI)*, pp. 910–916.

Korf, R. E., & Felner, A. (2002). Disjoint pattern database heuristics. *Artificial Intelligence*, *134*(1–2), 9–22.

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

Lawrence, S., & Giles, C. L. (1998). Searching the WWW. *Science*, *280*(5360), 98–100.

McCallum, A., Nigam, K., Rennie, J., & Seymore, K. (1999). Building domain-specific search engines with machine learning techniques. In *Proc. AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace*, pp. 28–39.

Menczer, F., Pant, G., Srinivasan, P., & Ruiz, M. (2001). Evaluating topic-driven web crawlers. In *Proceedings of SIGIR-01)*, pp. 241–249, New York. ACM Press.

Montgomery, D. C. (2001). *Design and Analysis of Experiments* (5 edition). John Wiley and Sons.

Mostow, J., & Prieditis, A. E. (1989). Discovering admissible heuristics by abstracting and optimizing: a transformational approach. In *Proceedings of IJCAI-89*, Vol. 1, pp. 701–707.

Najork, M., & Wiener, J. L. (1998). A technique for measuring the relative size and overlap of public web search engines. In *Proceedings of the Seventh International WWW Conference[WWW7]*, pp. 379–388.

Najork, M., & Wiener, J. L. (2001). Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th International WWW Conference*, pp. 114–118.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford Digital Library Technologies Project.

Pandurangan, G., Raghavan, P., & Upfal, E. (2002). Using PageRank to Characterize Web Structure. In *8th Annual International Computing and Combinatorics Conference (COCOON)*, pp. 330–339.

Pearl, J., & Kim, J. H. (1982). Studies in semi-admissible heuristics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *4*(4), 392–399.

Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics.. *12*(1–3), 117–141.

Rennie, J., & McCallum, A. K. (1999). Using reinforcement learning to spider the Web efficiently. In Bratko, I., & Dzeroski, S. (Eds.), *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pp. 335–343, Bled, SL. Morgan Kaufmann Publishers, San Francisco, US.

Russell, S. J., & Zilberstein, S. (1991). Composing real-time systems. In *Proceedings of IJCAI-91*, pp. 213–217, Sydney. Morgan Kaufmann.

Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2nd edition edition). Prentice-Hall, Englewood Cliffs, NJ.

Salton, G., & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management*, *24*(5), 513–523.

Schroedl, S. (2005). An improved search algorithm for optimal multiple-sequence alignment. *Journal of Artificial Intelligence Research*, *23*, 587–623.

Utgoff, P. (1988). An incremental ID3. In *Fifth International Conference on Machine Learning*, pp. 107–120. Morgan Kaufmann.

Yoshizumi, T., Miura, T., & Ishida, T. (2000). A * with partial expansion for large branching factor problems. In *AAAI/IAAI*, pp. 923–929.

Zhou, R., & Hansen, E. A. (2002). Multiple sequence alignment using anytime a*. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 975–976, Edmonton, Alberta, Canada.

Zhou, R., & Hansen, E. A. (2003). Sweep a*: Space-efficient heuristic search in partially ordered graphs. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 427–434, Sacramento, CA.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, *17*(3), 73–83.

Zilberstein, S., Charpillet, F., & Chassaing, P. (1999). Real-time problem-solving with contract algorithms. In *IJCAI*, pp. 1008–1015.