Efficient Goal-Directed Exploration

Yury Smirnov   Sven Koenig   Manuela M. Veloso   Reid G. Simmons
School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3891
{smir, skoenig, mmv, reids}@cs.cmu.edu

Abstract

If a state space is not completely known in advance, then search algorithms have to explore it sufficiently to locate a goal state and a path leading to it, performing therefore what we call goal-directed exploration. Two paradigms of this process are pure exploration and heuristic-driven exploitation: the former approaches explore the state space using only knowledge of the physically visited portion of the domain, whereas the latter approaches totally rely on heuristic knowledge to guide the search towards goal states. Both approaches have disadvantages: the first one does not utilize available knowledge to cut down the search effort, and the second one relies too much on the knowledge, even if it is misleading. We have therefore developed a framework for goal-directed exploration, called VECA, that combines the advantages of both approaches by automatically switching from exploitation to exploration on parts of the state space where exploitation does not perform well. VECA provides better performance guarantees than previously studied heuristic-driven exploitation algorithms, and experimental evidence suggests that this guarantee does not deteriorate its average-case performance.

Introduction

Classical AI search algorithms are concerned with finding action sequences that reach goal states from start states in completely known state spaces. Often, however, state spaces are not known in advance, and path finding algorithms need to gather information in the world to locate a goal state and a path leading to it. We call search problems of this kind **goal-directed exploration problems**. Examples include (A) mobile delivery robots that operate in initially unknown buildings, and (B) software agents that have to find World Wide Web pages of a given content by following links from their current page.

Two paradigms of goal-directed exploration are pure exploration and heuristic-driven exploitation: **Pure exploration approaches** explore the state space using only knowledge of the physically visited portion of the domain. **Heuristic-driven exploitation approaches**, on the other hand, totally rely on heuristic knowledge to guide the search towards goal states. Both approaches have disadvantages: the first one does not utilize available knowledge to cut down the search effort, and the second one relies too much on the knowledge, even if it is misleading.

We therefore introduce an application-independent framework for goal-directed exploration that can accommodate a wide variety of heuristic-driven exploitation algorithms and combines the advantages of both approaches: The **Variable Edge Cost Algorithm** (VECA) monitors whether the exploitation algorithm appears to perform poorly on some part of the state space, as indicated by the existence of actions that have been executed a large number of times. If so, VECA restricts the choices of the exploitation algorithm on that part of the state space, thus forcing it to explore the state space more. VECA provides a better performance guarantee than previously studied heuristic-driven exploitation algorithms, and misleading heuristic knowledge can never completely deteriorate its performance. A parameter of VECA determines when it starts to restrict the exploitation algorithm. This allows one to trade-off stronger performance guarantees (in case the heuristic knowledge is misleading) and more freedom of the exploitation algorithm (in case the quality of the heuristic knowledge is good). In its most stringent form, VECA's worst-case performance is guaranteed to be as good as that of the best uninformed goal-directed exploration algorithm. We present experimental evidence that suggests that VECA's performance guarantee does not greatly deteriorate the average-case performance of many previously studied heuristic-driven exploitation algorithms if they are used in conjunction with VECA; in many cases, VECA even improved their performance.

Goal-Directed Exploration Problems

We use the following notation: S denotes the finite set of states of the state space, $s_{start} \in S$ is the start state, and $G \subseteq S$ is the non-empty set of goal states. $A(s) \subseteq A$ is the set of actions that can be executed in $s \in S$. The execution of $a \in A(s)$ in s has cost $c(s, a) > 0$ and results in successor state $succ(s, a)$. The goal distance $gd(s)$ of s is the smallest cost with which a goal state can be reached from s. We assume that the goal distance of the start state is finite. We further assume that the state space is invertible ("undirected"), meaning that each action has an inverse (called the "twin" of the action). The weight of the state space is $weight = \sum_{s \in S} \sum_{a \in A(s)} c(s, a)$, the sum of the costs of all actions.

If $a \in A(s)$ is unexplored in s, then $c(s, a)$ and $succ(s, a)$ are unknown. To explore the action, the algorithm has to execute it in s. We assume that this reveals only $c(s, a)$ and $succ(s, a)$, but no additional information. Initially, heuristic knowledge about the effects of actions is available in form of estimates of the goal distances. Classical AI search algorithms attach heuristic values to states. This would force us to evaluate an unexplored action $a \in A(s)$ in s according to the heuristic value of s, since both $c(s, a)$ and $succ(s, a)$ are not yet known. We therefore attach heuristic values $h(s, a)$ to actions instead; they are estimates of $c(s,a)+gd(succ(s,a))$, the smallest cost of getting from s to a goal state when first executing a. If all $h(s,a)$ are zero, then the algorithm is uninformed.

The goal-directed exploration problem can now be stated as follows:

> **The Goal-Directed Exploration Problem**: Get an agent from s_{start} to a state in G if all actions are initially unexplored in all states, but heuristic estimates $h(s,a)$ are given.

We measure the performance of goal-directed exploration algorithms by the costs of their paths from the start state to a goal state. This performance measure is realistic, since the cost of executing actions often dominates the deliberation cost of goal-directed exploration algorithms.

Previous Approaches

In this section, we describe two approaches to goal-directed exploration: one pure exploration algorithm and one heuristic-based exploitation algorithm.[1]

A Pure Exploration Approach

Pure exploration approaches explore all actions. They have no notion of goal states and consequently do not use any prior knowledge to guide the search towards them. They can be used for goal-directed exploration, because they visit all states during their exploration, including the goal states. The following algorithm, whose exact origin is unclear, is an example. (Deng & Papadimitriou 1990) and (Korach, Kutten, & Moran 1990) stated it explicitly as a search algorithm, but it has been used earlier as part of proofs about Eulerian tours, for example in (Hierholzer 1873).

> **BETA ("Building a Eulerian Tour" Algorithm)**: Take unexplored actions whenever possible (ties can be broken arbitrarily). If all actions in the current state have been explored, execute the initial sequence of actions again, this time stopping at all states that have unexplored actions and apply the algorithm recursively from each such state.

BETA executes every action at most twice (Deng & Papadimitriou 1990). This implies the following theorem:

Theorem 1 *BETA solves any goal-directed exploration problem with a cost of* $\Theta(1) \times$ *weight (to be precise: with a cost of at most* $2 \times$ *weight).*

BETA does not make use of any prior knowledge to guide the search towards a goal state, although such knowledge can cut down the search effort and is often readily available. However, BETA provides a gold standard for the performance evaluation of goal-directed exploration algorithms, since no uninformed goal-directed exploration algorithm can do better in the worst case (Koenig & Smirnov 1996).

A Heuristic-Driven Exploitation Approach

Heuristic-driven exploitation approaches rely on heuristic knowledge to guide the search towards a goal state. The following algorithm is an example. It uses the A* algorithm to find a path to an unexplored action in the currently known part of the state space, moves the agent to that action, makes it execute the action, and repeats the process. This represents the idea behind algorithms such as Incremental Best-First Search (Pemberton & Korf 1992) and the Dynamic A* algorithm (Stentz 1995). The Learning Real-Time A* algorithm (Korf 1990), Prioritized Sweeping (Moore & Atkeson 1993b), and the navigation method described in (Benson & Prieditis 1992) are fast approximations of these algorithms.

> **AC-A* (Agent-Centered A* Algorithm)**: Consider all action sequences from the current state to an unexplored action. Select an action sequence with minimal cost from these sequences, where the cost of an action sequence is defined to be the sum of the cost of getting from the current state to s plus $h(s,a)$ (ties can be broken arbitrarily). Execute the action sequence and the unexplored action, and repeat the process until a goal state is reached.

AC-A* is very versatile: It can be used to search completely known, partially known, or completely unknown state spaces and is able to make use of knowledge that it acquires during the search. For example, if one informs it about the effects of some actions, it automatically utilizes this information during the remainder of its search.

The actions of AC-A* are optimal under the assumption that it has to explore only one more action. However, its behavior is not globally optimal: The worst-case performance of uninformed AC-A*, for example, increases faster than the weight of the state space and is thus worse than that of BETA (Koenig & Smirnov 1996):

Theorem 2 *The worst-case complexity of uninformed AC-A* is* $\Omega(\frac{\log|S|}{\log\log|S|}) \times$ *weight.*

Thus, the ability of AC-A* to use heuristic knowledge comes at the cost of a performance loss in the worst case if such knowledge is not available. Another disadvantage of AC-A* is that misleading heuristic values, even if they are consistent and thus admissible, can degrade its performance so much that it becomes worse than that of uninformed AC-A* and that of BETA (Smirnov *et al.* 1996). This does not imply, of course, that one should never use AC-A*. If AC-A* is totally informed, for example, it finds a goal state with cost $gd(s_{start})$ and thus cannot be outperformed by BETA or any other goal-directed exploration algorithm. The problem with AC-A* is that it takes the heuristic values at face value, even if its experience with them suggests that they should not be trusted.

[1] Other approaches have, for example, been discussed in theoretical computer science (Betke, Rivest, & Singh 1995; Blum, Raghavan, & Schieber 1991; Deng, Kameda, & Papadimitriou 1991), robotics (Rao *et al.* 1986; Lumelsky, Mukhopadhyay, & Sun 1990; Rao, Stoltzfus, & Iyengar 1991), and artificial intelligence (Thrun 1992; Sutton 1990; Moore & Atkeson 1993a).

Input:
a goal-directed exploration problem,
the value of VECA's parameter k (a non-negative, even integer),
and a heuristic-driven exploitation algorithm (to be used in step 3).

Basic-VECA uses three variables for each action a in state s: $count(s, a)$ keeps track of how often the action has been executed, $reserve(s, a)$ is its reserved VECA cost, and $cost(s, a)$ is its actual VECA cost.

1. Set $count(s, a) := cost(s, a) := 0$ for all $s \in S$ and $a \in A(s)$. Set $i := 0$ and $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. Consider all cycle-less action sequences starting at s and ending with an unexplored action. Select an action sequence with minimal VECA cost from these sequences, using the heuristic-driven exploitation algorithm to break ties.
4. Consider all actions in the action sequence, one after another. For each action a in the sequence, do:
   (a) Execute a in s. Let $a' \in A(s')$ be the twin of a in s.
   (b) Set $count(s, a) := count(s, a) + 1$.
   (c) If $count(s, a) + count(s', a') = 1$, then set $i := i + 1$ and afterwards $reserve(s, a) := reserve(s', a') := 2^{-i}$.
   (d) If $count(s, a) + count(s', a') \geq k$ and $cost(s, a) = 0$, then set $cost(s, a) := cost(s', a') := reserve(s, a)$.
   (e) If $count(s, a) + count(s', a') > k$, then set $cost(s, a) := \infty$.
   (f) Set $s := succ(s, a)$.
5. Go to step 2.

Alternatively, step 3 can be replaced by:

3' Consider all cycle-less action sequences starting at s and ending with an action whose VECA cost is zero. Select an action sequence with minimal VECA cost from these sequences, using the heuristic-driven exploitation algorithm to break ties.

Figure 1: The Basic-VECA Framework

Our Approach: The VECA Framework

We have developed a framework for goal-directed exploration, called the Variable Edge Cost Algorithm (VECA), that can accommodate a wide variety of heuristic-driven exploitation algorithms (including AC-A*). VECA relies on the exploitation algorithm and thus on the heuristic values until they prove to be misleading. To this end, it monitors the behavior of the exploitation algorithm and uses a parameter k to determine when the freedom of the exploitation algorithm should get restricted. If an action and its twin together have been executed k times or more, VECA restricts the choices of the exploitation algorithm on that part of the state space, thus forcing it to explore the state space more. As a result, VECA switches gradually from exploitation to exploration and relies less and less on misleading heuristic values.

We describe VECA in two stages. We first discuss a simple version of VECA, called Basic-VECA, that assumes that executing an action also identifies its twin. Later, we drop this assumption. Basic-VECA is described in Figure 1. It maintains a cost for each action that is different from the cost $c(s, a)$. These VECA costs guide the search. Initially, all of them are zero. Whenever Basic-VECA executes a pair of twin actions for the first time (i.e. it executes one of the two actions for the first time and has not yet executed the other one), it reserves a VECA cost for them, that will later become their VECA cost. The first pair of twin actions gets a cost of 1/2 reserved, the second pair a cost of 1/4, the third pair a cost of 1/8, and so on. Basic-VECA assigns the reserved cost to both twin actions when it executes the pair for the kth time (or, if $k = 0$, when it executes the pair for the first time). Whenever the pair is executed again, Basic-VECA assigns the executed action (but not its twin) an infinite VECA cost, which effectively removes it. The VECA costs are used as follows: Basic-VECA always chooses a sequence of actions with minimal VECA cost that leads from its current state to an unexplored action or, alternatively, to an action with zero VECA cost. The exploitation algorithm is used to break ties. Initially, all VECA costs are zero and there are lots of ties to break. The more actions Basic-VECA assigns non-zero VECA costs to, the fewer ties there are and the less freedom the exploitation algorithm has.

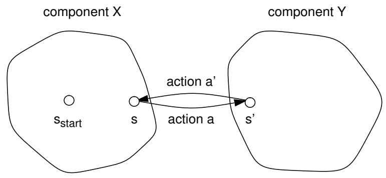

Figure 2: A Simple Example State Space

To gain an intuitive understanding of the behavior of Basic-VECA, consider a simple case, namely a tree-shaped state space, and assume that Basic-VECA uses step 3. Figure 2 shows a pair of twin edges, a and a', that connect two components of the tree, X and Y. The exploitation algorithm can execute a freely until it and its twin a' together have been executed a total of k times. Then, Basic-VECA assigns both actions the same positive VECA cost. At this point in time, the agent is located in X (the component that contains the start state), since k is even and the agent alternates between both components. If Y does not contain any more unexplored actions, neither a nor a' will be executed again. Otherwise there is a point in time when Basic-VECA executes a again to reach one of those unexplored actions. When this happens, Basic-VECA prevents the exploitation algorithm from leaving Y until all actions in Y have been explored (this restriction of the freedom of the exploitation algorithm constitutes a switch from exploitation to more exploration): Because Y can only be entered by executing a, this action was executed before any action in Y. Consequently, its reserved VECA cost, which is also its current VECA cost, is larger than the reserved VECA cost of any action in Y. The reserved VECA costs are exponentially decreasing: $2^{-i} > \sum_{j>i} 2^{-j}$ for all finite $i, j \geq 0$. Thus, any action sequence that does not leave Y has a smaller VECA cost than any action sequence that does, and Basic-VECA cannot leave Y until all of Y's actions have been explored. When Basic-VECA has finally left Y, the VECA costs of a and a' are infinite, but there is no need to enter or exit Y again.

In general, Basic-VECA executes every pair of twin actions a total of at most $k+2$ times before it finds a goal state (Smirnov *et al.* 1996). This implies the following theorem:

Theorem 3 *Under the assumption that executing an action also identifies its twin, Basic-VECA, with even parameter* $k \geq 0$*, solves any goal-directed exploration problem with a*

Input:
a goal-directed exploration problem,
the value of VECA's parameter k (a non-negative, even integer),
and a heuristic-driven exploitation algorithm (to be used in step 3).

VECA uses four variables for each action a in state s: $count(s, a)$ keeps track of how often the action has been executed, $reserve(s, a)$ is its reserved VECA cost, $cost(s, a)$ is its actual VECA cost, and $euler(s, a)$ remembers whether it has already been executed as part of step 4(e).

1. Set $count(s, a) := cost(s, a) := euler(s, a) := 0$ for all $s \in S$ and $a \in A(s)$. Set $i := 0$ and $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. Consider all cycle-less action sequences starting at s and ending with an unexplored action. Select an action sequence with minimal VECA cost from these sequences, using the heuristic-driven exploitation algorithm to break ties.
4. Consider all actions in the action sequence, one after another. For each action a in the sequence, do:
   (a) Execute a in s.
   (b) Set $count(s, a) := count(s, a) + 1$.
   (c) If $count(s, a) = 1$ and the twin of a in s is not yet known, then set $i := i + 1$ and afterwards $reserve(s, a) := 2^{-i}$.
   (d) If $count(s, a) = 1$ and the twin of a in s is known, then let $a' \in A(s')$ be the twin of a in s. If $count(s', a') = 0$, then set $i := i + 1$ and afterwards $reserve(s, a) := 2^{-i}$, else set $reserve(s, a) := reserve(s', a')$.
   (e) If $count(s, a) \geq k$ and the twin of a in s is not yet known, then do:
      i. Set $s'' := succ(s, a)$.
      ii. Select an action a'' in s'' with $euler(s'', a'') = 0$. If there is no such action, then go to step 4(f) (comment: it holds that $s'' = succ(s, a)$).
      iii. Execute a'' in s''.
      iv. Set $euler(s'', a'') := 1$.
      v. Set $s'' := succ(s'', a'')$.
      vi. Go to step 4(e)ii.
   (f) If the twin of a in s is known, then let $a' \in A(s')$ be the twin of a in s and do:
      i. If $count(s, a) + count(s', a') \geq k$ and $cost(s, a) = 0$, then set $cost(s, a) := cost(s', a') := reserve(s, a)$.
      ii. If $count(s, a) + count(s', a') > k$, then set $cost(s, a) := \infty$.
   (g) Set $s := succ(s, a)$.
5. Go to step 2.

Alternatively, step 3 can be replaced by:

3' Consider all cycle-less action sequences starting at s and ending with an action whose VECA cost is zero. Select an action sequence with minimal VECA cost from these sequences, using the heuristic-driven exploitation algorithm to break ties.

Figure 3: The VECA Framework

cost of $\Theta(1) \times$ weight (to be precise: with a cost of at most $(k/2 + 1) \times$ weight).

A larger k allows the exploitation algorithm to maintain its original behavior longer, whereas a smaller k forces it earlier to explore the state space more. The smaller the value of k, the better the performance guarantee of Basic-VECA. If $k = 0$, for example, Basic-VECA severely restricts the freedom of the exploitation algorithm and behaves like chronological backtracking. In this case, it executes every action at most once (for a total cost of *weight*), no matter how misleading its heuristic knowledge is or how bad the choices of the exploitation algorithm are. No uninformed goal-directed exploration algorithm can do better in the worst case even if executing an action also identifies its twin. However, if the heuristic values are not misleading, a small value of k can force the exploitation algorithm to explore the state space unnecessarily. Thus, a stronger performance guarantee might come at the expense of a decrease in average-case performance. The experiments in the section on "Experimental Results" address this issue.

VECA is very similar to Basic-VECA, see Figure 3. In contrast to Basic-VECA, however, it does not assume that executing an action identifies its twin. This complicates the algorithm somewhat: First, the twin of an action might not be known when VECA reserves a VECA cost for the pair. This requires an additional amount of bookkeeping. Second, the twin of an action might not be known when VECA wants to assign it the VECA cost. In this case, VECA is forced to identify the twin: step 4(e) explores all actions in the state that contains the twin (including the twin) and returns to that state. This procedure is executed only rarely for larger k, since it is almost never the case that the twin of an action that has already been executed k times is not yet known. Because of this step, though, VECA can potentially execute any action one more time than Basic-VECA, which implies the following theorem (Smirnov *et al.* 1996):

Theorem 4 *VECA, with even parameter $k \geq 0$, solves any goal-directed exploration problem with a cost of $\Theta(1) \times$ weight (to be precise: with a cost of at most $(k/2 + 2) \times$ weight).*

For $k = 0$, VECA executes every action at most twice. Thus, its worst-case performance is at most $2 \times$ *weight* and equals the worst-case performance of BETA. No uninformed goal-directed exploration algorithm can do better in the worst case if executing an action does not identify its twin.

Implementation

Since the VECA costs are exponentially decreasing and the precision of numbers on a computer is limited, Basic-VECA and VECA cannot be implemented exactly as described. Instead, we represent action sequences as lists that contain the current VECA costs of their actions in descending order. All action sequences of minimal VECA cost then have the smallest lexicographic order. Since this relationship continues to hold if we replace the VECA costs of the actions with their exponent (for example, we use -3 if the VECA cost of an action is $1/8 = 2^{-3}$), we can now use small integers instead of exponentially decreasing real values, and steps 3 and 3' can be implemented efficiently using a simple modification of Dijkstra's algorithm in conjunction with priority lists.

Experimental Results

We augment our theoretical worst-case analysis with an experimental average-case analysis, because the worst-case complexity of an algorithm often does not predict its average-case performance well. The task that we studied was finding goals in mazes. The mazes were constructed by first generating an acyclic maze of size 64×64 and then randomly removing 32 walls. The action costs corresponded to the travel distances; the shortest distance between two junctions counted as one unit. We randomly created ten mazes with start location (62,62), goal location (0,0), diameters between 900 and 1000 units, and goal distances of the start state between 650 and 750 units. For every goal-directed exploration algorithm, we performed ten trials in each of

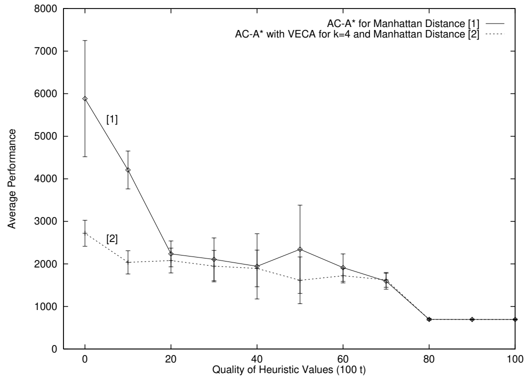

Figure 4: AC-A* with and without VECA

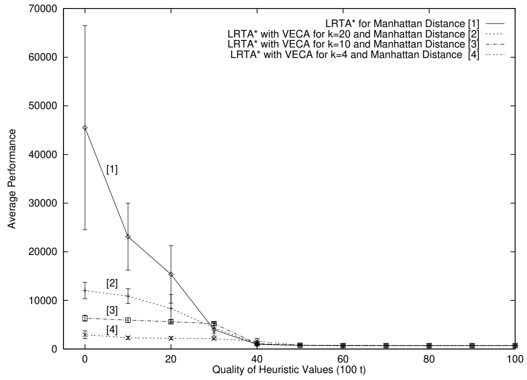

Figure 5: LRTA* with and without VECA

the ten mazes (with ties broken randomly). We repeated the experiments for different values of VECA's parameter k and for heuristic values $h(s, a)$ of different quality. The heuristic values were generated by combining the goal distance $gd(s)$ of a state s with its Manhattan distance $mh(s)$, the sum of the x and y distance from s to the goal state. A parameter $t \in [0, 1]$ determined how misleading the heuristic values were; a smaller t implied a lower quality:

$$h(s, a) = c(s, a) + t \times gd(succ(s, a)) + (1 - t) \times mh(succ(s, a)).$$

Here, we report the results for two heuristic-driven exploitation algorithms, namely AC-A* and Learning Real-Time A* (LRTA*) with look-ahead one (Korf 1990). We integrated these algorithms into the VECA framework as follows: AC-A* was used with step 3 of VECA and broke ties among action sequences according to their total cost (see the definition of AC-A*). LRTA* was used with step 3' of VECA and broke ties according to the heuristic value of the last action in the sequence. These ways of integrating AC-A* and LRTA* with VECA are natural extensions of the stand-alone behavior of these algorithms. If the heuristic values are misleading, AC-A* is more efficient than LRTA* (this is to be expected, since AC-A* deliberates more between action executions). As a result, VECA was able to improve the average-case performance of LRTA* more than it could improve the performance of AC-A*.

Figure 4 shows the average-case performance of AC-A* with and without VECA (including 95 percent confidence intervals). The x axis shows $100 \times t$, our measure for the quality of the heuristic values, and the y axis shows the average travel distance from the start to the goal, averaged over all 100 trials. All graphs tend to decrease for increasing t, implying that the performance of the algorithms increased with the quality of the heuristic values (as expected). AC-A* without VECA was already efficient and executed each action only a small number of times. Thus, VECA did not change the behavior of AC-A* when k was large. For example, for $k = 10$, the behavior of AC-A* with VECA (not shown in the figure) was the same as the behavior of AC-A* without VECA. The graphs for $k = 4$ suggest that AC-A* with VECA outperforms AC-A* without VECA if the heuristic values are of bad quality and that it remains competitive even for heuristic values of higher quality.

Figure 5 shows the average-case performance of LRTA* with and without VECA. As before, the performance of the algorithms increased with the quality of the heuristic values. The graphs show that, most of the time, LRTA* with VECA outperformed or tied with LRTA* without VECA. For misleading heuristic values (small t), LRTA* with VECA worked the better, the smaller the value of VECA's parameter k was. However, for only moderately misleading heuristic values (t between 0.3 and 0.5), a larger value of k achieved better results and LRTA* without VECA even outperformed LRTA* with VECA if k was small. This was the case, because the heuristic values guided the search better and a small value of k forced LRTA* to explore the state space too much.

We obtained similar results for both experiments when we generated the heuristic values differently, for example as the combination of the goal distance and the Euclidean distance or the goal distance and the larger coordinate difference between a state and the goal. We also performed experiments with other heuristic-driven exploitation algorithms, such as biased random walks or an algorithm that expands the states in the same order as the A* algorithm. Since both of these algorithms are inefficient to begin with, VECA was able to achieve large performance improvements for misleading heuristic values.

Extensions

In this paper, we have assumed that a goal-directed exploration algorithm learns only the effect of the executed action, but not the effects of any other actions. However, Basic-VECA and VECA can be used unchanged in environments in which action executions provide more information (such as, for example, complete information about the effects of all actions in the vicinity of the agent) and Theorems 3 and 4 continue to hold.

Conclusion

We introduced an application-independent framework for goal-directed exploration, called VECA, that addresses a variety of search problems in the same framework and provides good performance guarantees. VECA can accommodate a wide variety of exploitation strategies that use heuristic knowledge to guide the search towards a goal state. It monitors whether the heuristic-driven exploitation algorithm appears to perform poorly on some part of the state space. If so, it forces the exploitation algorithm to explore the state space more. This combines the advantages of pure exploration approaches and heuristic-driven exploitation approaches: VECA is able to utilize heuristic knowledge, but provides a better performance guarantee than previously studied heuristic-driven exploitation algorithms (such as the AC-A* algorithm): VECA's worst-case performance is always linear in the weight of the state space. Thus, while misleading heuristic values do not help it to find a goal state faster, they cannot completely deteriorate its performance either. A parameter of VECA determines when it starts to restrict the choices of the heuristic-driven exploitation algorithm. This allows one to trade-off stronger performance guarantees (in case the heuristic knowledge is misleading) and more freedom of the exploitation algorithm (in case the quality of the heuristic knowledge is good). In its most stringent form, VECA's worst-case performance is guaranteed to be as good as that of BETA, the best uninformed goal-directed exploration algorithm. Our experiments suggest that VECA's performance guarantee does not greatly deteriorate the average-case performance of many previously studied heuristic-driven exploitation algorithms if they are used in conjunction with VECA; in many cases, VECA even improved their performance. In future work, we will study the influence of domain properties on the performance of goal-directed exploration algorithms in the VECA framework.

Acknowledgements

This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, the Advanced Research Projects Agency under grant number F33615-93-1-1330 and the National Science Foundation under grant number IRI-9502548. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

References

Benson, G., and Prieditis, A. 1992. Learning continuous-space navigation heuristics in real time. In *Proceedings of the Conference on Simulation of Adaptive Behavior (SAB): From Animals to Animats*.

Betke, M.; Rivest, R.; and Singh, M. 1995. Piecemeal learning of an unknown environment. *Machine Learning* 18(2/3).

Blum, A.; Raghavan, P.; and Schieber, B. 1991. Navigation in unfamiliar terrain. In *Proceedings of the Symposium on Theory of Computing (STOC)*, 494–504.

Deng, X., and Papadimitriou, C. 1990. Exploring an unknown graph. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 355–361.

Deng, X.; Kameda, T.; and Papadimitriou, C. 1991. How to learn an unknown environment. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*.

Hierholzer, C. 1873. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen* 6:30–32.

Koenig, S., and Smirnov, Y. 1996. Graph learning with a nearest neighbor approach. In *Proceedings of the Conference on Computational Learning Theory (COLT)*.

Korach, E.; Kutten, S.; and Moran, S. 1990. A modular technique for the design of efficient distributed leader finding algorithms. *ACM Transactions on Programming Languages and Systems* 12(1):84–101.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

Lumelsky, V.; Mukhopadhyay, S.; and Sun, K. 1990. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation* 6(4):462–472.

Moore, A., and Atkeson, C. 1993a. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. In *Advances in Neural Information Processing Systems 6 (NIPS)*.

Moore, A., and Atkeson, C. 1993b. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

Pemberton, J., and Korf, R. 1992. Incremental path planning on graphs with cycles. In *Proceedings of the AI Planning Systems Conference (AIPS)*, 179–188.

Rao, N.; Iyengar, S.; Jorgensen, C.; and Weisbin, C. 1986. Robot navigation in an unexplored terrain. *Journal of Robotic Systems* 3(4):389–407.

Rao, N.; Stoltzfus, N.; and Iyengar, S. 1991. A "retraction" method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation* 7(5):699–707.

Smirnov, Y.; Koenig, S.; Veloso, M. M.; and Simmons, R. G. 1996. Goal-directed exploration in the VECA framework. Technical report, School of Computer Science, Carnegie Mellon University.

Stentz, A. 1995. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1652–1659.

Sutton, R. 1990. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning (ICML)*, 216–224.

Thrun, S. 1992. Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University.