AI Magazine Volume 25 Number 2 (2004) (© AAAI)

Incremental Heuristic Search in AI

Sven Koenig, Maxim Likhachev, Yaxin Liu, and David Furcy

■ Incremental search reuses information from previous searches to find solutions to a series of similar search problems potentially faster than is possible by solving each search problem from scratch. This is important because many AI systems have to adapt their plans continuously to changes in (their knowledge of) the world. In this article, we give an overview of incremental search, focusing on LIFELONG PLANNING A*, and outline some of its possible applications in AI.

Overview

It is often important that searches be fast. AI has developed several ways of speeding up searches by trading off the search time and the cost of the resulting path, which includes using inadmissible heuristics (Pohl 1973, 1970) and search with limited look ahead (Korf 1990; Ishida and Korf 1991; Koenig 2001), which is also called real-time or agent-centered search. In this article, we discuss a different way of speeding up searches, namely, incremental search. *Incremental search* is a search technique for continual planning (or, synonymously, replanning, plan reuse, and lifelong planning) that reuses information from previous searches to find solutions to a series of similar search problems potentially faster than is possible by solving each search problem from scratch. Different from other ways of speeding up searches, it can guarantee to find the shortest paths. Notice that the terminology is unfortunately somewhat problematic because the term *incremental search* in computer science also refers to both online search and search with limited look ahead (Pemberton and Korf 1994).

Most of the research on search has studied how to solve one-time search problems. However, many AI systems have to adapt their plans continuously to changes in the world or changes of their models of the world, for example, because the actual situation turns out to be slightly different from the one initially assumed or because the situation changes over time. In these cases, the original plan might no longer apply or might no longer be good, and one thus needs to replan for the new situation (desJardins et al. 1999). Similarly, one needs to solve a series of similar search problems if one wants to perform a series of what-if analyses or if the costs of planning operators, their preconditions, or their effects change over time because they are learned or refined.

In these situations, most search algorithms replan from scratch, that is, solve the new search problem independently of the old ones. However, this approach can be inefficient in large domains with frequent changes and, thus, limit the responsiveness of AI systems or the number of what-if analyses that they can perform. Fortunately, the changes to the search problems are usually small. A robot, for example, might have to replan when it detects a previously unknown obstacle, a traffic routing system might have to replan when it learns about a new traffic jam, and a decision support system for marine oil spill containment might have to replan when the wind direction changes. This property suggests that a complete recomputation of the best plan for the new search problem is unnecessary because some of the previous search results can be reused, which is what incremental search does.

Incremental search solves *dynamic shortest-path problems,* where shortest paths have to be found repeatedly as the topology of a graph or its edge costs change (Ramalingam and Reps 1996b). The idea of incremental search is old. For example, an overview article about shortest-path algorithms from 1984 already cites several incremental search algorithms, including several ones published in the late 1960s

(Deo and Pang 1984). Since then, additional incremental search algorithms have been suggested in the algorithms literature (Ausiello et al. 1991; Even and Gazit 1985; Even and Shiloach 1981; Feuerstein and Marchetti-Spaccamela 1993; Franciosa, Frigioni, and Giaccio 2001; Frigioni, Marchetti-Spaccamela, and Nanni 1996; Goto and Sangiovanni-Vincentelli 1978; Italiano 1988; Klein and Subramanian 1993; Lin and Chang 1990; Rohnert 1985; Spira and Pan 1975) and, to a much lesser degree, the AI literature (Al-Ansari 2001; Edelkamp 1998). They differ in their assumptions, for example, whether they solve single-source or all-pairs shortest-path problems, which performance measure they use, when they update the shortest paths, which kinds of graph topologies and edge costs they apply to, and how the graph topology and edge costs are allowed to change over time (Frigioni, Marchetti-Spaccamela, and Nanni 1998). If arbitrary sequences of edge insertions, deletions, or weight changes are allowed, then the dynamic shortest-path problems are called *fully dynamic shortest-path problems* (Frigioni, Marchetti-Spaccamela, and Nanni 2000).

The idea of incremental search has also been pursued in AI for problems other than path finding. For example, a variety of algorithms have been developed for solving constraint-satisfaction problems (Dechter and Dechter 1988; Verfaillie and Schiex 1994) or constraint logic programming problems (Miguel and Shen 1999) where the constraints change over time. In this article, however, we study incremental search only in the context of dynamic shortest-path problems, where it has not been studied extensively in AI.

We believe that four achievements are necessary to make incremental search more popular in AI. First, one needs to devise more powerful incremental search algorithms than those that currently exist. Second, one needs to study their properties more extensively, both analytically and experimentally, to understand their strengths and limitations better. Third, one needs to demonstrate that they apply to AI applications and compare them to other search algorithms for these applications to demonstrate that they indeed have advantages over them. Finally, the AI community needs to be made more aware of incremental search. This article addresses the last issue so that more researchers can address the first three. It describes one particular incremental search algorithm and its potential applications and then discusses its potential advantages and limitations.

Uninformed Incremental Search

We now discuss one particular way of solving fully dynamic shortest-path problems. As an example, we use route planning in known eight-connected gridworlds with cells whose traversability changes over time. They are either traversable (with cost one) or untraversable. The *route-planning problem* is to repeatedly find a shortest path between two given cells of the gridworld, knowing both what the topology of the gridworld is and which cells are currently traversable. It can be solved with conventional search, such as breadth-first search, by finding a shortest path every time some edge costs change. Conventional search typically does not reuse information from previous searches. The following example, however, illustrates the potential advantage of reusing information from previous searches.

Consider the gridworlds shown in figure 1. The original gridworld is shown in figure 1a, and the changed gridworld is shown in figure 1b. Only two blockages have moved, resulting in four cells with changed blockage status. The figure shows the shortest paths in both cases under the assumption that every move has cost one. The shortest path changed because two cells on the original shortest path became untraversable.

The length of a shortest path from the start cell to a cell is called its *start distance.* Once the start distances of all cells are known, one can easily trace back a shortest path from the start cell to the goal cell by always greedily decreasing the start distance, starting at the goal cell. The start distances are shown in each traversable cell of the original and changed gridworlds (figures 1a and 1b). Those cells whose start distances in the changed gridworld (figure 1b) are different from the corresponding cells in the original gridworld; they are shaded gray in figure 1b. Even though large parts of the shortest path have changed, less than a quarter of the start distances have changed. Thus, in such cases, there is a potential advantage to recalculating only those start distances that have changed, which is an argument for caching the start distances rather than the shortest path itself.

Caching the start distances is basically what the incremental search algorithm DynamicSWSF-FP (Ramalingam and Reps 1996a) does.[1] DynamicSWSF-FP was originally developed in the context of parsing theory and theoretical computer science. It uses a clever way of identifying the start distances that have not changed and recalculates only the ones that have changed. Consequently, it performs best in situations where only a small number of start distances change.

A. Original Eight-Connected Gridworld

S_{start}	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	2	3	4	5	6	7	8	9
3									
4	4	5	6	7		12	12	12	13
5	5			7		11	11	12	13
6	6	6	7	8		10	11	12	13
				8	9	10	11	12	13
12	11	10	9	9			11		13
12	11	10	10	10			12	12	S_{goal}

B. Changed Eight-Connected Gridworld

S_{start}	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	2	3	4	5	6	7	8	9
3									9
4	4	5	6	7		12	11	10	10
5	5					11	11	11	11
6	6	6	7	8		10	11	12	12
		7		8	9	10	11	12	13
9	8	8	8	9			11		13
9	9	9	9	9			12		S_{goal}

Figure 1. Simple Gridworld.

Consider the gridworld in figure 2 to understand how it operates. The start cell is A3. Assume that one is given the values in the left gridworld, and it is claimed that they are equal to the correct start distances. There are at least two different approaches to verify this property. One approach is to perform a complete search to find the start distances and compare them to the given values. Another approach is to check the definition of the start distances, namely, that the value of the start cell is zero, and the value of every other cell is equal to the minimum over all neighboring cells of the value of the neighboring cell plus the cost of getting from the neighboring cell to the cell in question, which is indeed the case. For example, the value of cell B1 should be the minimum of the values of cells A0, A1, A2, and C1 plus one. Thus, the values are indeed equal to the correct start distances. Both approaches need about the same run time to confirm this property. Now assume that cell D1 becomes untraversable, as shown in the right gridworld of figure 2, and thus, the costs of all edges into the cell become infinity, and it is claimed that the values in the cells remain equal to the cor-

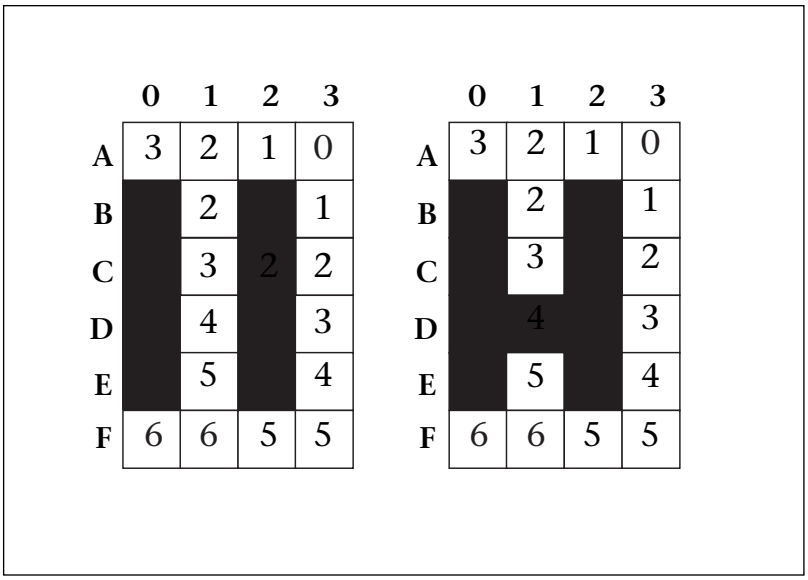

Figure 2. An Example.

rect start distances. Again, there are at least two different approaches to verify this property. One approach is again to perform a complete search to find the start distances and compare them to the given values. The second approach is again to check that the value of the start cell is zero and that the value of every other cell is equal to the minimum over all neighboring cells of the value of the neighboring cell plus the cost of getting from the neighboring cell to the cell in question. Because the values remain unchanged, each cell continues to have this property unless its neighbors have changed. Thus, one needs to check only whether the cells close to changes in the gridworld continue to have this property, that is, cells C1 and E1. It turns out that cell C1 continues to have this property, but cell E1 does not. Thus, not all values are equal to the correct start distances (which does not mean, of course, that all cells but E1 have correct start distances). The second approach now needs less run time than the first one. Furthermore, the second approach provides a starting point for replanning, namely, cell E1, because one needs to work on the cells that do not have this property, moving outward from the starting point. The principle behind the second approach is the main idea behind DynamicSWSF-FP. It shares this idea with other incremental search approaches, including some that apply to constraint satisfaction (Dechter and Dechter 1988).

Informed Incremental Search

One way of speeding up searches is to use incremental search. Another way of speeding up searches is to use heuristic search. The question arises whether incremental and heuristic search can be combined. Many of the start distances that have changed in the example in figure 1 are irrelevant for finding a shortest path from the start cell to the goal cell and thus do not need to be recalculated. Examples are the cells in the lower left corner of the gridworld. Thus, there is a potential advantage to using heuristics to avoid having to recalculate irrelevant start distances.

To summarize, there are two different ways to decrease the search effort of breadth-first search for finding the start distances for the changed gridworld (figure 1b):

First, *incremental search algorithms,* such as DynamicSWSF-FP (Ramalingam and Reps 1996a), find shortest paths for series of similar search problems potentially faster than is possible by solving each search problem from scratch (complete search) because they do not recompute those start distances that have not changed.

Second, *heuristic search algorithms,* such as A* (Nilsson 1971), use heuristic knowledge in the form of approximations of the goal distances to focus the search and find shortest paths for search problems faster than uninformed search because they do not compute those start distances that are irrelevant for finding a shortest path from the start cell to the goal cell.

Consequently, we developed a search algorithm that combines incremental and heuristic search, namely, LPA* (LIFELONG PLANNING A*) (Koenig and Likhachev 2002b). We call it *lifelong planning* in analogy to lifelong learning (Thrun 1998) because it reuses information from previous searches. LPA* repeatedly finds shortest paths from a given start vertex to a given goal vertex on arbitrary known finite graphs (not just gridworlds) whose edge costs increase or decrease over time (which can also be used to model edges or vertices that are added or deleted). The pseudocode of the simplest version of LPA* reduces to a version of A* that breaks ties among vertices with the same *f*-value in favor of smaller *g*-values when used to search from scratch and to DynamicSWSF-FP when used with uninformed heuristics. In fact, it differs from DynamicSWSF-FP only in the calculation of the priorities for the vertices in the priority queue (line {01} in the pseudocode of figure 3). It is unoptimized and needs consistent heuristics (Pearl 1985). We have also developed more sophisticated versions of LPA* that are optimized (for example, recalculate the

various values much more efficiently than the simple version), can work with inadmissible heuristics, and can break ties among vertices with the same *f*-value in favor of larger *g*-values. These changes make LPA* more complex.

Replanning with LPA* can best be understood as transforming the A* search tree of the old search problem to the A* search tree of the new search problem, which results in some computational overhead because parts of the old A* search tree need to be undone. It also results in computational savings because other parts of the old A* search tree can be reused. The larger the overlap between the old and new A* search trees, the more efficient replanning with LPA* is compared to using A* to create the new search tree from scratch. For example, LPA* is likely more efficient in situation 1a of figure 4 than in situation 1b.[2] Experimental results show that LPA* is the more efficient, for example, the less the graph has changed (see experiment 1) and the closer the edges with changed cost are to the goal of the search (see experiment 3). The computational savings can dominate the computational overhead, and LPA* then replans faster than A*. In the worst case, however, no search algorithm is more efficient than a complete search from scratch (Nebel and Koehler 1995), and LPA* can be less efficient than A*, which can happen if the overlap of the old and new A* search trees is small.

The simplicity of LPA* allows us to prove a number of properties about it, including its termination, correctness, efficiency in terms of vertex expansions, and similarity to A*, which makes it easy to understand, analyze, and extend, for example, to nondeterministic domains (Likhachev and Koenig 2003). We can prove, for example, that the first search of LPA* expands the vertices in the same order as a version of A* that breaks ties among vertices with the same *f*-value in favor of smaller *g*-values. LPA* expands every vertex at most twice and does not expand many vertices at all because it does not expand vertices whose values were already equal to their start distances (efficiency as a result of incremental search) or whose previous and current *f*-values are larger than the *f*-value of the goal vertex (efficiency as a result of heuristic search). Details can be found in Koenig, Likhachev, and Furcy (2004).

More information on incremental search in general can be found in Frigioni et al. (2000), and more information on LPA* can be found in Koenig, Likhachev, and Furcy (2004) (this article is an introductory version of it).

S denotes the finite set of vertices of the graph. $succ(s) \subseteq S$ denotes the set of successors of vertex $s \in S$. Similarly, $pred(s) \subseteq S$ denotes the set of predecessors of vertex $s \in S$. $0 < c(s, s') \leq \infty$ denotes the cost of moving from vertex s to vertex $s' \in succ(s)$. LPA* always determines a shortest path from a given start vertex $s_{start} \in S$ to a given goal vertex $s_{goal} \in S$, knowing both the topology of the graph and its current edge costs. The heuristics need to be nonnegative and consistent.

LPA* maintains estimates $g(s)$ and $rhs(s)$ of the start distance of each vertex s. LPA* also maintains a priority queue that contains exactly the vertices s with $g(s) \neq rhs(s)$. Their priorities are pairs, the first component of which is similar to an f-value of A* and the second one of which is similar to a g-value of A*. The priorities are compared to according to a lexicographic ordering. For example, a priority $[k_1; k_2]$ is less than or equal to a priority $[k_1'; k_2']$ iff either $k_1 < k_1'$ or ($k_1 = k_1'$ and $k_2 \leq k_2'$). U.TopKey() returns the smallest priority of all vertices in priority queue U. (If U is empty, then U.TopKey() returns $[\infty; \infty]$.) U.Pop() deletes the vertex with the smallest priority in priority queue U and returns the vertex. U.Insert(s, k) inserts vertex s into priority queue U with priority k. Finally, U.Remove(s) removes vertex s from priority queue U.

```
procedure CalculateKey(s)
{01} return [min(g(s), rhs(s)) + h(s); min(g(s), rhs(s))];

procedure Initialize()
{02} U = ∅;
{03} for all s ∈ S rhs(s) = g(s) = ∞;
{04} rhs(s_start) = 0;
{05} U.Insert(s_start, [h(s_start); 0]);

procedure UpdateVertex(u)
{06} if (u ≠ s_start) rhs(u) = min_{s'∈pred(u)}(g(s') + c(s', u));
{07} if (u ∈ U) U.Remove(u);
{08} if (g(u) ≠ rhs(u)) U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath()
{09} while (U.TopKey()<CalculateKey(s_goal) OR rhs(s_goal) ≠ g(s_goal))
{10}     u = U.Pop();
{11}     if (g(u) > rhs(u))
{12}         g(u) = rhs(u);
{13}         for all s ∈ succ(u) UpdateVertex(s);
{14}     else
{15}         g(u) = ∞;
{16}         for all s ∈ succ(u) ∪ {u} UpdateVertex(s);

procedure Main()
{17} Initialize();
{18} forever
{19}     ComputeShortestPath();
{20}     Wait for changes in edge costs;
{21}     for all directed edges (u, v) with changed edge costs
{22}         Update the edge cost c(u, v);
{23}         UpdateVertex(v);
```

Figure 3. LIFELONG PLANNING A* *(simple version).*

Experimental Evaluation

We now perform experiments in small gridworlds to better understand the advantages and limitations of LPA* (Koenig, Likhachev, and Furcy (2004). We compare an optimized version of LPA* against a version of A* that breaks ties among vertices with the same *f*-value in favor of vertices with larger *g*-values because doing so tends to result in a smaller number of vertex expansions than breaking ties in the opposite direction (although tie breaking turns out not to make a big difference in our gridworlds). All priority queues were implemented as binary heaps.[3]

We use four connected gridworlds with directed edges between adjacent cells. We generate 100 gridworlds. The start and goal cells are

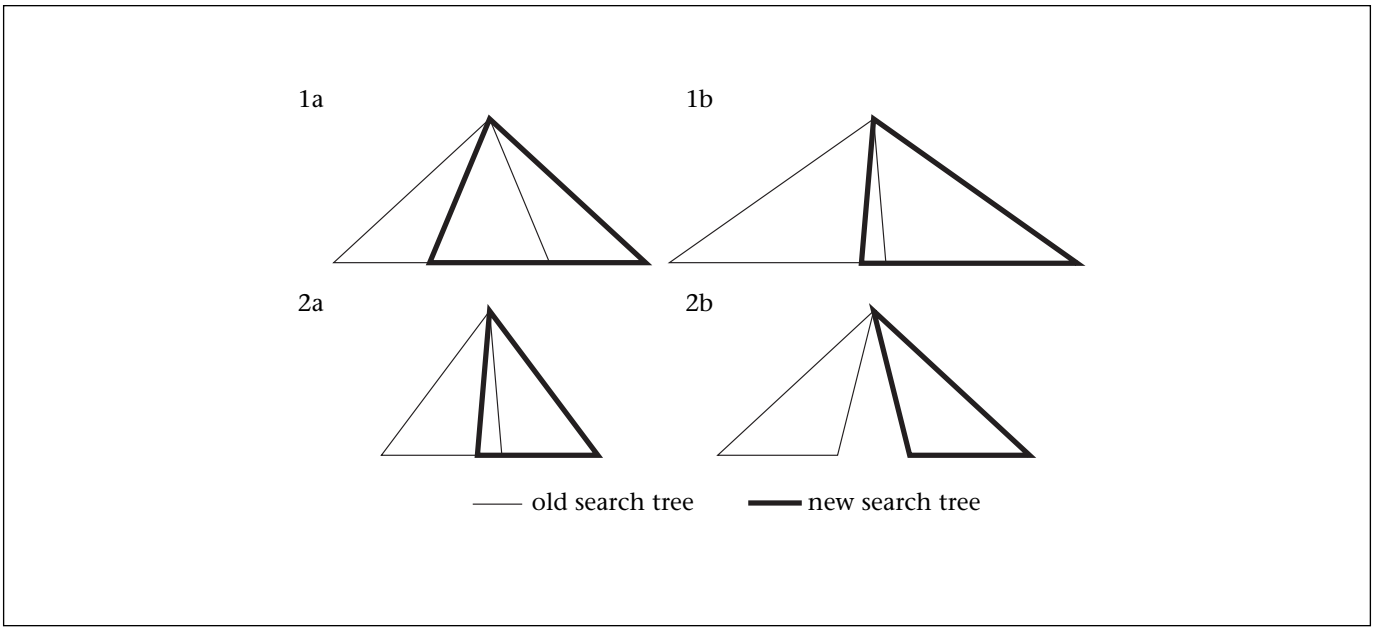

Figure 4. Old and New Search Trees.

drawn with uniform probability for each gridworld. All edge costs are either one or two with uniform probability. We then change each gridworld 500 times in a row by selecting a given number of edges and reassigning them random costs. We report the probability that the cost of the shortest path changes to ensure that the edge cost changes indeed change the shortest path sufficiently often. A probability of 33.9 percent, for example, means that the cost of the shortest path changes on average after 2.96 planning episodes. We use the *Manhattan distances* as heuristics for the cost of a shortest path between two cells, that is, the sum of the difference of their *x*- and *y*-coordinates. For each experiment (tables 1, 2, and 3), we report the run time (in milliseconds) averaged over all first planning episodes (#1) and over all planning episodes (#2), run on a PENTIUM 1.7-megahertz PC. We also report the *speedup* of LPA* over A* in the long run (#3), that is, the ratio of the run times of A* and LPA* averaged over all planning episodes. The first search of LPA* tends to be slower than that of A* because it expands more states and needs more time for each state expansion. During the subsequent searches, however, LPA* often expands fewer states than A* and is thus faster than A*. We therefore also report the replanning episode after which the average total run time of LPA* is smaller than that of A* (#4), in other words, the number of replanning episodes that are necessary for one to prefer LPA* over A*. For example, if this number is two, then LPA* solves one planning problem and two replanning problems together faster than A*. Additional experiments are reported in Koenig, Likhachev, and Furcy (2004).

Experiment 1

In the first experiment, the size of the gridworlds is 101 by 101. We change the number of edges that get assigned random costs before each planning episode. Table 1 shows our experimental results. The smaller the number of edges that get assigned random costs, the less the search space changes and the larger the advantage of LPA* in our experiments. The average run time of the first planning episode of LPA* tends to be larger than that of A*, but the average run time of the following planning episodes tends to be so much smaller (if the number of edges that get assigned random costs is sufficiently small) that the number of replanning episodes that are necessary for one to prefer LPA* over A* is one.

Experiment 2

In the second experiment, the number of edges that get assigned random costs before each planning episode is 0.6 percent. We change the size of the square gridworlds. Table 2 shows our experimental results. The smaller the gridworlds, the larger the advantage of LPA* in our experiments, although we were not able to pre-

edge cost changes	path cost changes	A* #1 and #2	LPA* #1	#2	#3	#4
0.2%	3.0%	0.299	0.386	0.029	10.370×	1
0.4%	7.9%	0.336	0.419	0.067	5.033×	1
0.6%	13.0%	0.362	0.453	0.108	3.344×	1
0.8%	17.6%	0.406	0.499	0.156	2.603×	1
1.0%	20.5%	0.370	0.434	0.174	2.126×	1
1.2%	24.6%	0.413	0.476	0.222	1.858×	1
1.4%	28.7%	0.468	0.539	0.282	1.657×	1
1.6%	32.6%	0.500	0.563	0.332	1.507×	1
1.8%	32.1%	0.455	0.497	0.328	1.384×	1
2.0%	33.8%	0.394	0.433	0.315	1.249×	1

Table 1. Experiment 1.

maze size	path cost changes	A* #1 and #2	LPA* #1	#2	#3	#4
51 × 51	7.3%	0.077	0.098	0.015	5.032×	1
76 × 76	10.7%	0.201	0.258	0.050	3.987×	1
101 × 101	13.0%	0.345	0.437	0.104	3.315×	1
126 × 126	16.2%	0.690	0.789	0.220	3.128×	1
151 × 151	17.7%	0.933	1.013	0.322	2.900×	1
176 × 176	21.5%	1.553	1.608	0.564	2.753×	1
201 × 201	22.9%	1.840	1.898	0.682	2.696×	1

Table 2. Experiment 2.

dict this effect. This insight is important because it implies that LPA* does not scale well in our gridworlds (although part of this effect could be the result of the fact that more edges get assigned random costs as the size of the gridworlds increases; this time is included in the run time averaged over all planning episodes). We, therefore, devised the third experiment.

Experiment 3

In the third experiment, the number of edges that get assigned random costs before each planning episode is again 0.6 percent. We change both what the size of the square gridworlds is and how close the edges that get assigned random costs are to the goal cell. Eighty percent of these edges leave cells that are close to the goal cell. Table 3 shows our experimental results. Now, the advantage of LPA* no longer decreases with the size of the gridworlds. The closer the edge cost changes are to the goal cell, the larger the advantage of LPA* is in our experiments, as predicted earlier. This insight is important because it suggests using LPA* when most of the edge cost changes are close to the goal cell. We use this property when we apply LPA* to mobile robotics and control (see later discussion).

Although these experiments give us some insight into the behavior of LPA*, we need to improve our understanding of when to prefer incremental search over alternative search algorithms and which incremental search algorithm to use. The main criteria for choosing a search algorithm are its memory consumption and its run time.

With respect to memory, incremental search needs memory for information from past

80% of edge cost changes are ≤ 25 cells away from the goal

maze size	path cost changes	A* #1 and #2	LPA* #1	LPA* #2	LPA* #3	LPA* #4
51 × 51	13.5%	0.084	0.115	0.014	6.165×	1
76 × 76	23.9%	0.189	0.245	0.028	6.661×	1
101 × 101	33.4%	0.295	0.375	0.048	6.184×	1
126 × 126	42.5%	0.696	0.812	0.084	8.297×	1
151 × 151	48.5%	0.886	0.964	0.114	7.808×	1
176 × 176	55.7%	1.353	1.450	0.156	8.683×	1
201 × 201	59.6%	1.676	1.733	0.202	8.305×	1

80% of edge cost changes are ≤ 50 cells away from the goal

maze size	path cost changes	A* #1 and #2	LPA* #1	LPA* #2	LPA* #3	LPA* #4
51 × 51	8.6%	0.086	0.115	0.017	5.138×	1
76 × 76	15.7%	0.190	0.247	0.039	4.822×	1
101 × 101	23.2%	0.304	0.378	0.072	4.235×	1
126 × 126	31.3%	0.702	0.812	0.130	5.398×	1
151 × 151	36.2%	0.896	0.959	0.173	5.166×	1
176 × 176	44.0%	1.372	1.458	0.242	5.664×	1
201 × 201	48.3%	1.689	1.742	0.313	5.398×	1

80% of edge cost changes are ≤ 75 cells away from the goal

maze size	path cost changes	A* #1 and #2	LPA* #1	LPA* #2	LPA* #3	LPA* #4
76 × 76	12.1%	0.196	0.250	0.047	4.206×	1
101 × 101	17.5%	0.306	0.391	0.088	3.499×	1
126 × 126	26.0%	0.703	0.818	0.175	4.012×	1
151 × 151	28.8%	0.893	0.972	0.225	3.978×	1
176 × 176	36.8%	1.370	1.438	0.319	4.301×	1
201 × 201	40.1%	1.728	1.790	0.408	4.236×	1

Table 3. Experiment 3.

searches. LPA*, for example, needs to remember the previous search tree. This property of LPA* tends not to be a problem for gridworlds, but the search trees of other search problems are often so large that they do not completely fit into memory. In this case, it might be possible to combine incremental search with memory-limited search algorithms such as RBFS (Korf 1993) or SMA* (Russell 1992), but this work is for the future.

With respect to run time, our experiments have demonstrated that LPA* expands fewer vertices than A*, for example, when only a few edge costs change, and these edge costs are close to the goal vertex. These situations need to be characterized better. LPA* also needs more time for each vertex expansion than A*. This time disadvantage depends on low-level implementation and machine details, such as the instruction set of the processor, the optimizations performed by the compiler, and the data structures used for the priority queues; it is thus hard to characterize. For example, when the number of edges that get assigned random costs was 0.2 percent in experiment 1, the number of heap percolates of A* was 8213.04, but the number of heap percolates of LPA* was only 297.30. This result makes it difficult to determine whether there is a benefit to incremental search and, if so, how to quantify it. For example, one can decrease the speedup of LPA* over A* by using buckets to implement the priority queues rather than heaps, even though this approach is more complicated. We implemented A* with buckets and a simple FIFO tie-breaking strategy within buckets (which speeds it up by more than a factor of two) but left the implementation of LPA* unchanged. In this

case, LPA* needed more than one replanning episode in experiment 1 to outperform A* if the number of edges that got reassigned random costs before each planning episode was less than 1.0 percent and did not outperform A* at all if the number of edges that got reassigned random costs before each planning episode was 1.0 percent or more. Therefore, we are only willing to conclude from our experiments that incremental heuristic search is a promising technology that needs to be investigated further. In general, the trade-off between the number of vertex expansions and the time needed for each vertex expansion might benefit incremental search algorithms that are less sophisticated than LPA* and, thus, expand more vertices but with less time for each vertex expansion. One might also be able to develop incremental search algorithms that apply only to special cases of graphs (such as gridworlds) and, thus, are faster but not as versatile as LPA*.

Applications

Our gridworld experiments suggest that incremental search can be beneficial for route planning in traffic or computer networks, where the congestion and, thus, the optimal routes change over time. These applications are similar to our gridworld experiments. However, a variety of areas in AI could also potentially benefit from incremental search. In the following subsections, we discuss some of the possible applications, many (but not all) of which involve gridworlds. We also discuss the current state of the art, including opportunities for future research on incremental search.

Symbolic Planning (HSP)

Symbolic planning is the most obvious application of incremental search. Interestingly, incremental search can be used not only to solve a series of similar symbolic (STRIPS-style) planning problems but also single symbolic planning problems.

One-time planning: Heuristic search-based planners solve symbolic planning problems. They were introduced by McDermott (1996) and Bonet, Loerincs, and Geffner (1997) and have become very popular. In its default configuration, HSP 2.0 (Bonet and Geffner 2000), for example, uses weighted A* searches with inadmissible heuristics to perform forward searches in the space of world states to find a path from the start state to a goal state. However, the calculation of the heuristics is time consuming because HSP 2.0 calculates the heuristic of each state that it encounters during the search by solving a relaxed search problem. Consequently, the calculation of the heuristics comprises about 80 percent of its run time (Bonet and Geffner 2001).

HSP 2.0 thus repeatedly solves relaxed search problems as it calculates the heuristics. The relaxed search problems that it solves to find the heuristics of two states are similar if the two states are similar, and two states whose heuristics it calculates consecutively are often similar because they are both children of the same parent in the A* search tree. Thus, incremental search can solve a relaxed search problem by reusing information from the calculation of the previous relaxed search problem. Our PINCH (prioritized, incremental heuristics calculation) algorithm (Liu, Koenig, and Furcy 2002), for example, is based on DYNAMICSWSF-FP and speeds up the run time of HSP 2.0 by as much as 80 percent in several domains, and in general, the amount of savings grows with the size of the domains, allowing HSP 2.0 to solve larger search problems with the same limit on its run time and without changing its heuristics or overall operation.

Continual planning: So far, we have described how incremental search can solve single symbolic planning problems. However, planning researchers realized a long time ago that one often does not need to solve just one symbolic planning problem but, rather, a series of similar symbolic planning problems. Examples of practical significance include the aeromedical evacuation of injured people in crisis situations (Kott, Saks, and Mercer 1999) and air campaign planning (Myers 1999). Replanning is necessary in these cases, for example, when a landing strip at an airfield becomes unusable. Planning researchers have therefore studied replanning and plan reuse. *Replanning* attempts to retain as many plan steps of the previous plan as possible. Plan *reuse* does not have this requirement. (We do not make this distinction and, thus, use the term *replanning* throughout the text.) Examples include case-based planning, planning by analogy, plan adaptation, transformational planning, planning by solution replay, repair-based planning, and learning search-control knowledge. These search algorithms have been used as part of systems such as CHEF (Hammond 1990), GORDIUS (Simmons 1988), LS-ADJUST-PLAN (Gerevini and Serina 2000), MRL (Koehler 1994), NOLIMIT (Veloso 1994), PLEXUS (Alterman 1988), PRIAR (Kambhampati and Hendler 1992), and SPA (Hanks and Weld 1995).

HSP 2.0 with weight one and consistent heuristics finds plans with minimal plan-execution cost. If HSP 2.0 solves a series of similar symbolic planning problems, then it can use LPA* instead of A* to replan faster, resulting in

the SHERPA (speedy heuristic search-based) replanner (Koenig, Furcy, and Bauer 2002) for consistent heuristics. A difference between SHERPA and the other replanners described earlier is that SHERPA not only remembers the previous plans but also remembers the previous plan-construction processes. Thus, it has more information available for replanning than even PRIAR, which stores plans together with explanations of their correctness, or NOLIMIT, which stores plans together with substantial descriptions of the decisions that resulted in the solution. Another difference between SHERPA and the other replanners is that its plan-execution cost is as good as the plan-execution cost achieved by search from scratch. Thus, incremental search can be used for plan reuse if the plan-execution cost of the resulting plan is important, but its similarity to the previous plans is not.

Inadmissible heuristics allow HSP 2.0 to solve search problems in large state spaces by trading off run time and the plan-execution cost of the resulting plan. SHERPA uses LPA* with consistent heuristics. Although we have extended LPA* to use inadmissible heuristics and still guarantee that it expands every vertex at most twice, it turns out to be difficult to make incremental search more efficient than search from scratch with the same inadmissible heuristics, although we have had success in special cases. This difficulty can be explained as follows: The larger the heuristics are, the narrower the A* search tree and, thus, the more efficient A* is. However, the narrower the A* search tree, the more likely it is that the overlap between the old and new A* search trees is small and, thus, the less efficient LPA* is. For example, situations 2a and 2b in figure 4 correspond to situations 1a and 1b respectively, except that the old and new search trees are narrower and, thus, overlap less.

Mobile Robotics and Games (Path Planning)

Mobile robots often have to replan quickly as the world or their knowledge of it changes. Examples include both physical robots and computer-controlled robots (or, more generally, computer-controlled characters) in computer games. Efficient replanning is especially important for computer games because they often simulate a large number of characters, and their other software components, such as the graphics generation, already place a high demand on the processor. In the following paragraphs, we discuss two cases where the knowledge of a robot changes because its sensors acquire more information about the initially unknown terrain as it moves around.

Goal-directed navigation in unknown terrain: Planning with the free-space assumption is a popular solution to the goal-directed navigation problem where a mobile robot has to move in initially unknown terrain to given goal coordinates. For example, the characters in popular combat games such as TOTAL ANNIHILATION, AGE OF EMPIRES, and WARCRAFT have to move autonomously in initially unknown terrain to user-specified coordinates. *Planning with the free-space assumption* always plans a shortest path from its current coordinates to the goal coordinates under the assumption that the unknown terrain is traversable. When it observes obstacles as it follows this path, it enters them into its map and then repeats the procedure, until it eventually reaches the goal coordinates, or all paths to them are untraversable.

To implement this navigation strategy, the robot needs to replan shortest paths whenever it detects that its current path is untraversable. Several ways of speeding up the searches have been proposed in the literature (Barbehenn and Hutchinson 1995; Ersson and Hu 2001; Huiming et al. 2001; Podsedkowski et al. 2001; Tao et al. 1997; Trovato 1990). FOCUSSED DYNAMIC A* (D*) (Stentz 1995) is probably the most popular solution and has been extensively used on real robots, such as outdoor high-mobility multipurpose wheeled vehicles (HMMWVs) (Hebert, McLachlan, and Chang 1999; Matthies et al. 2000; Stentz and Hebert 1995; Thayer et al. 2000), and studied theoretically (Koenig, Tovey, and Smirnov 2003). We believe that D* is the first truly incremental heuristic search algorithm. It resulted in a new application for incremental search and a major advance in robotics. LPA* and D* share similarities. For example, we can combine LPA* with ideas from D* to apply it to moving robots, resulting in D* LITE (Koenig and Likhachev 2002a). D* LITE and D* implement the same navigation strategy and are about equally fast, but D* LITE is algorithmically simpler and, thus, easy to understand, analyze, and extend. Both search algorithms search from the goal coordinates toward the current coordinates of the robot. Because the robot usually observes obstacles close to its current coordinates, the changes are close to the goal of the search, which makes incremental search efficient, as predicted earlier.

Mapping: *Greedy mapping* is a popular solution to the problem of mapping unknown terrain (Koenig, Tovey, and Haliburton 2001; Romero, Morales, and Sucar 2001; Thrun et al. 1998). The robot always plans a shortest path from its current coordinates to a closest patch of terrain with unknown traversability until the terrain is mapped.

To implement this navigation strategy, the robot needs to replan shortest paths whenever it observes new obstacles. Both D* and D* LITE can be used unchanged to implement greedy mapping, although their advantage over A* is much smaller for mapping than for goal-directed navigation in unknown terrain (Likhachev and Koenig 2002).

Machine Learning (Reinforcement Learning)

Reinforcement learning is learning from rewards and penalties that can be delayed (Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998). Reinforcement learning algorithms, such as Q-LEARNING (Watkins and Dayan 1992) or online versions of value iteration (Barto, Bradtke, and Singh 1995), often use dynamic programming to update state or state-action values and are then similar to real-time search (Ishida and Korf 1991; Koenig 2001; Korf 1990). The order of the value updates determines how fast they can propagate information through the state space, which has a substantial effect on their efficiency. The DYNA-Q environment (Sutton and Barto 1998) has been used by various researchers to study ways of making reinforcement learning more efficient by ordering their value updates. PRIORITIZED SWEEPING (Moore and Atkeson 1993) and QUEUE-DYNA (Peng and Williams 1993) are, for example, reinforcement learning algorithms that resulted from this research. They concentrate the value updates on those states whose values they change most.

Incremental search can order the value updates in an even more systematic way and uses concepts related to concepts from reinforcement learning. For example, LPA* performs dynamic programming and implicitly uses the Bellman (1957) equations. It is currently unclear how incremental search can be applied to minimizing the expected (discounted) plan-execution cost in nondeterministic domains, the typical objective of reinforcement learning. However, we have extended LPA* from minimizing the plan-execution cost in deterministic domains to minimizing the worst-case plan-execution cost in nondeterministic domains, resulting in MINIMAX LPA* (Likhachev and Koenig 2003), which we believe to be the first incremental heuristic minimax search algorithm. It applies to goal-directed reinforcement learning for minimizing the worst-case plan-execution cost. Although this is the only application of incremental search to reinforcement learning that has been identified to date, it suggests that ideas from incremental search can potentially be used to reduce the number of values that reinforcement learning algorithms need to update.

Control (PARTI-GAME Algorithm)

State spaces of control problems are often continuous and sometimes high dimensional. The *PARTI-GAME algorithm* (Moore and Atkeson 1995) finds control policies that move an agent in such domains from given start coordinates to given goal coordinates. It is popular because it is simple and efficient and applies to a broad range of control problems such as path planning for mobile robots and robot arms (Al-Ansari and Williams 1999; Araujo and de Almeida 1998, 1997; Kollmann et al. 1997). To solve these problems, one can first discretize the domains and then use conventional search to find plans that move the agent from its current coordinates to the goal coordinates. However, uniform discretizations can prevent one from finding a plan if they are too coarse grained (for example, because the resolution prevents one from noticing small gaps between obstacles) and result in large state spaces that cannot be searched efficiently if they are too fine grained. The PARTI-GAME algorithm solves this dilemma by starting with a coarse discretization and refines it during execution only when and where it is needed (for example, around obstacles), resulting in a nonuniform discretization. To implement the PARTI-GAME algorithm, the agent needs to find a plan with minimal worst-case plan-execution cost whenever it has refined its model of the domain. Several ways of speeding up the searches with incremental search have been proposed in the literature. Al-Ansari (2001), for example, proposed an uninformed incremental search algorithm that restores the priority queue of a minimax search algorithm (similar to Dijkstra's algorithm) to the vertices it had immediately before the changed edge costs made it behave differently, and we proposed a combination of MINIMAX LPA* and D* LITE (Likhachev and Koenig 2003).

Conclusions

Incremental search reuses information from previous searches to find solutions to a series of similar search problems potentially faster than is possible by solving each search problem from scratch. Although incremental search is currently not used much in AI, this article demonstrated that there are AI applications that might benefit from incremental search. It also demonstrated that we need to improve our understanding of incremental search, including when to prefer incremental search over alternative search algorithms and which incre-

mental search algorithm to use. For example, the search spaces of incremental search methods (for example, for computer games) are often small, and thus, their scaling properties are less important than implementation and machine details. At the same time, it is difficult to compare them using proxies, such as the number of vertex expansions, if they perform very different basic operations. We therefore suggest that AI researchers study incremental search in more depth in the coming years to understand it better; develop new algorithms; evaluate its potential for real-world applications; and in general, determine whether incremental search is an important technique in the tool box of AI.

Acknowledgments

This research was performed when the authors were at Georgia Institute of Technology. Thanks to Peter Yap, Rob Holte, and Jonathan Schaeffer for interesting insights into the behavior of LPA*. Thanks also to Anthony Stentz and Craig Tovey for helpful discussions and to Colin Bauer for implementing some of our ideas. The Intelligent Decision-Making Group is partly supported by National Science Foundation awards to Sven Koenig under contracts IIS-9984827, IIS-0098807, and ITR/AP-0113881. Yaxin Liu was supported by an IBM Ph.D. fellowship. The views and conclusions contained in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies, or the U.S. government.

Notes

1. DynamicSWSF-FP, as originally stated, searches from the goal vertex to all other vertices and, thus, maintains estimates of the goal distances rather than the start distances. We, however, use it to search from the start vertex to all other vertices. Also, to calculate a shortest path from the start vertex to the goal vertex, not all distances need to be known. To make DynamicSWSF-FP more efficient, we changed its termination condition so that it stops immediately after it has found a shortest path from the start vertex to the goal vertex.

2. To be more precise, it is not only important that the trees are similar, but most start distances of its nodes have to be the same as well. Thus, it is insufficient, for example, that the tree remains unchanged if most of the start distances of its nodes change.

3. The code of our implementations is available at www-rcf.usc.edu/~skoenig/fastreplanning.html.

References

Al-Ansari, M. 2001. Efficient Reinforcement Learning in Continuous Environments. Ph.D. dissertation, College of Computer Science, Northeastern University.

Al-Ansari, M., and Williams, R. 1999. Robust, Efficient, Globally Optimized Reinforcement Learning with the PARTI-GAME Algorithm. In *Advances in Neural Information Processing Systems, Volume 11,* 961–967. Cambridge, Mass.: MIT Press.

Alterman, R. 1988. Adaptive Planning. *Cognitive Science* 12(3): 393–421.

Araujo, R., and de Almeida, A. 1998. Map Building Using Fuzzy Art and Learning to Navigate a Mobile Robot on an Unknown World. In Proceedings of the International Conference on Robotics and Automation, Volume 3, 2554–2559. Washington, D.C.: IEEE Computer Society.

Araujo, R., and de Almeida, A. 1997. Sensor-Based Learning of Environment Model and Path Planning with a Nomad 200 Mobile Robot. In Proceedings of the International Conference on Intelligent Robots and Systems, Volume 2, 539–544. Washington, D.C.: IEEE Computer Society.

Ausiello, G.; Italiano, G.; Marchetti-Spaccamela, A.; and Nanni, U. 1991. Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms* 12(4): 615–638.

Barbehenn, M., and Hutchinson, S. 1995. Efficient Search and Hierarchical Motion Planning by Dynamically Maintaining Single-Source Shortest Paths Trees. *IEEE Transactions on Robotics and Automation* 11(2): 198–214.

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to Act Using Real-Time Dynamic Programming. *Artificial Intelligence* 73(1): 81–138.

Bellman, R. 1957. *Dynamic Programming.* Princeton, N.J.: Princeton University Press.

Bonet, B., and Geffner, H. 2001. Planning as Heuristic Search. *Artificial Intelligence* 129(1): 5–33.

Bonet, B., and Geffner, H. 2000. HEURISTIC SEARCH PLANNER 2.0. *AI Magazine* 22(3): 77–80.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A Robust and Fast Action Selection Mechanism. In Proceedings of the Fourteenth National Conference on Artificial Intelligence, 714–719. Menlo Park, Calif.: American Association for Artficial Intelligence.

Dechter, R., and Dechter, A. 1988. Belief Maintenance in Dynamic Constraint Networks. In Proceedings of the Seventh National Conference on Artificial Intelligence, 37–42. Menlo Park, Calif.: American Association for Artificial Intelligence.

Deo, N., and Pang, C. 1984. Shortest-Path Algorithms: Taxonomy and Annotation. *Networks* 14:275–323.

desJardins, M.; Durfee, E.; Ortiz, C.; and Wolverton, M. 1999. A Survey of Research in Distributed, Continual Planning. *AI Magazine* 20(4): 13–22.

Edelkamp, S. 1998. Updating Shortest Paths. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence,* 655–659. Chichester, U.K.: Wiley.

Ersson, T., and Hu, X. 2001. Path Planning and Navigation of Mobile Robots in Unknown Environments. In Proceedings of the International Conference on Intelligent Robots and Systems, 858–864.

Washington, D.C.: IEEE Computer Society.

Even, S., and Gazit, H. 1985. Updating Distances in Dynamic Graphs. *Methods of Operations Research* 49: 371–387.

Even, S., and Shiloach, Y. 1981. An Online Edge- Deletion Problem. *Journal of the ACM* 28(1): 1–4.

Feuerstein, E., and Marchetti-Spaccamela, A. 1993. Dynamic Algorithms for Shortest Paths in Planar Graphs. *Theoretical Computer Science* 116(2): 359–371.

Franciosa, P.; Frigioni, D.; and Giaccio, R. 2001. Semidynamic Breadth-First Search in Digraphs. *Theoretical Computer Science* 250(1–2): 201–217.

Frigioni, D.; Marchetti-Spaccamela, A.; and Nanni, U. 2000. Fully Dynamic Algorithms for Maintaining Shortest-Path Trees. *Journal of Algorithms* 34(2): 251–281.

Frigioni, D.; Marchetti-Spaccamela, A.; and Nanni, U. 1998. Semidynamic Algorithms for Maintaining Single-Source Shortest-Path Trees. *Algorithmica* 22(3): 250–274.

Frigioni, D.; Marchetti-Spaccamela, A.; and Nanni, U. 1996. Fully Dynamic Output Bounded Single Source Shortest Path Problem. In Proceedings of the Seventh Annual Symposium on Discrete Algorithms, 212–221. New York: Association of Computing Machinery.

Gerevini, A., and Serina, I. 2000. Fast Plan Adaptation through Planning Graphs: Local and Systematic Search Techniques. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems,* eds. S. Chien, S. Kambhampati, and C. A. Knoblock, 112–121. Menlo Park, Calif.: AAAI Press.

Goto, S., and Sangiovanni-Vincentelli, A. 1978. A New Shortest-Path Updating Algorithm. *Networks* 8(4): 341–372.

Hammond, K. 1990. Explaining and Repairing Plans That Fail. *Artificial Intelligence* 45(1–2): 173–228.

Hanks, S., and Weld, D. 1995. A Domain-Independent Algorithm for Plan Adaptation. *Journal of Artificial Intelligence Research* 2:319–360.

Hebert, M.; McLachlan, R.; and Chang, P. 1999. Experiments with Driving Modes for Urban Robots. In Proceedings of the SPIE Mobile Robots, 140–149. Bellingham, Wash.: International Society for Optical Engineering.

Huiming, Y.; Chia-Jung, C.; Tong, S.; and Qiang, B. 2001. Hybrid Evolutionary Motion Planning Using Follow Boundary Repair for Mobile Robots. *Journal of Systems Architecture* 47(7): 635–647.

Ishida, T., and Korf, R. 1991. Moving Target Search. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 204–210. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Italiano, G. 1988. Finding Paths and Deleting Edges in Directed Acyclic Graphs. *Information Processing Letters* 28(1): 5–11.

Kaelbling, L.; Littman, M.; and Moore, A. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4:237–285.

Kambhampati, S., and Hendler, J. 1992. A Validation-Structure-Based Theory of Plan Modification and Reuse. *Artificial Intelligence* 55(2): 193–258.

Klein, P., and Subramanian, S. 1993. A Fully Dynamic Approximation Scheme for All-Pairs Shortest Paths in Planar Graphs. In *Algorithms and Data Structures: Third Workshop, WADS'93,* 443–451. Lecture Notes in Computer Science 709. New York: Springer-Verlag.

Koehler, J. 1994. Flexible Plan Reuse in a Formal Framework. In *Current Trends in AI Planning,* eds. *C. Bäckström and E. Sandewall,* 171–184. Amsterdam, The Netherlands: IOS.

Koenig, S. 2001. Agent-Centered Search. *AI Magazine* 22(4): 109–131.

Koenig, S., and Likhachev, M. 2002a. D* LITE. In Proceedings of the Eighteenth National Conference on Artificial Intelligence, 476–483. Menlo Park, Calif.: American Association for Artificial Intelligence.

Koenig, S., and Likhachev, M. 2002b. INCREMENTAL A*. In *Advances in Neural Information Processing Systems 14,* eds. T. Dietterich, S. Becker, and S. Ghahramani, 1539–1546. Cambridge, Mass.: MIT Press.

Koenig, S.; Furcy, D.; and Bauer, C. 2002. Heuristic Search-Based Replanning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems,* eds. M. Ghallab, J. Hertzberg, and P. Traverso, 294–301. Menlo Park, Calif.: AAAI Press.

Koenig, S.; Likhachev, M.; and Furcy, D. 2004. LIFELONG PLANNING A*. *Artificial Intelligence Journal* 155:93–146.

Koenig, S.; Tovey, C.; and Halliburton, W. 2001. Greedy Mapping of Terrain. In Proceedings of the IEEE International Conference on Robotics and Automation, 3594–3599. Washington, D.C.: IEEE Computer Society.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance Bounds for Planning in Unknown Terrain. *Artificial Intelligence* 147(1–2): 253–279.

Kollmann, J.; Lankenau, A.; Buhlmeier, A.; Krieg-Bruckner, B.; and Rofer, T. 1997. Navigation of a Kinematically Restricted Wheelchair by the PARTI-GAME Algorithm. In Proceedings of the Society for the Study of Artificial Intelligence and the Simulation of Behavior 1997 Workshop on Spatial Reasoning in Mobile Robots and Animals, 35–44. Technical Report, UMCS-97-4-1, Department of Computer Science, University of Manchester.

Korf, R. 1993. Linear-Space Best-First Search. *Artificial Intelligence* 62(1): 41–78.

Korf, R. 1990. Real-Time Heuristic Search. *Artificial Intelligence* 42(2-3): 189–211.

Kott, A.; Saks, V.; and Mercer, A. 1999. A New Technique Enables Dynamic Replanning and Rescheduling of Aeromedical Evacuation. *AI Magazine* 20(1): 43–53.

Likhachev, M., and Koenig, S. 2003. Speeding Up the PARTIGAME Algorithm. In *Advances in Neural Information Processing Systems 15,* eds. S. Becker, S. Thrun, and K. Obermayer. Cambridge, Mass.: MIT Press.

Likhachev, M., and Koenig, S. 2002. Incremental Replanning for Mapping. In Proceedings of the International Conference on Intelligent Robots and Systems, 667–672. Washington, D.C.: IEEE Computer Society.

Lin, C., and Chang, R. 1990. On the Dynamic Shortest-Path Problem. *Journal of Information Processing* 13(4): 470–476.

Liu, Y.; Koenig, S.; and Furcy, D. 2002. Speeding Up the Calculation of Heuristics for Heuristic Search-Based Planning. In Proceedings of the Eighteenth National Conference on Artificial Intelligence, 484–491. Menlo Park, Calif.: American Association for Artificial Intelligence.

McDermott, D. 1996. A Heuristic Estimator for Means-Ends Analysis in Planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems,* ed. Brian Drabble, 142–149. Menlo Park, Calif.: AAAI Press.

Matthies, L.; Xiong, Y.; Hogg, R.; Zhu, D.; Rankin, A.; Kennedy, B.; Hebert, M.; Maclachlan, R.; Won, C.; Frost, T.; Sukhatme, G.; McHenry, M.; and Goldberg, S. 2000. A Portable, Autonomous, Urban Reconnaissance Robot. In Proceedings of the International Conference on Intelligent Autonomous Systems. Amsterdam, The Netherlands: IOS.

Miguel, I., and Shen, Q. 1999. Extending FCSP to Support Dynamically Changing Problems. In Proceedings of IEEE International Fuzzy Systems Conference, 1615–1620. Washington, D.C.: IEEE Computer Society.

Moore, A., and Atkeson, C. 1995. The PARTIGAME Algorithm for Variable Resolution Reinforcement Learning in Multidimensional State Spaces. *Machine Learning* 21(3): 199–233.

Moore, A., and Atkeson, C. 1993. PRIORITIZED SWEEPING: Reinforcement Learning with Less Data and Less Time. *Machine Learning* 13(1): 103–130.

Myers, K. 1999. CPEF: A Continuous Plan-

ning and Execution Framework. *Artificial Intelligence* 20(4): 63–69.

Nebel, B., and Koehler, J. 1995. Plan Reuse versus Plan Generation: A Theoretical and Empirical Analysis. *Artificial Intelligence* 76(1–2): 427–454.

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. New York: McGraw-Hill.

Pearl, J. 1985. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, Mass.: Addison-Wesley.

Pemberton, J., and Korf, R. 1994. Incremental Search Algorithms for Real-Time Decision Making. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems,* ed. K. Hammond, 140–145. Menlo Park, Calif.: AAAI Press.

Peng, J., and Williams, R. 1993. Efficient Learning and Planning within the DYNA Framework. *Adaptive Behavior* 1(4): 437–454.

Podsedkowski, L.; Nowakowski, J.; Idzikowski, M.; and Vizvary, I. 2001. A New Solution for Path Planning in Partially Known or Unknown Environments for Nonholonomic Mobile Robots. *Robotics and Autonomous Systems* 34:145–152.

Pohl, I. 1973. The Avoidance of (Relative) Catastrophe, Heuristic Competence, Genuine Dynamic Weighting and Computational Issues in Heuristic Problem Solving. In Proceedings of the International Conference on Intelligent Robots and Systems, 20–23. Washington, D.C.: IEEE Computer Society.

Pohl, I. 1970. First Results on the Effect of Error in Heuristic Search. In *Machine Intelligence, Volume 5,* eds. B. Meltzer and D. Michie, 219–236. New York: American Elsevier.

Ramalingam, G., and Reps, T. 1996a. An Incremental Algorithm for a Generalization of the Shortest-Path Problem. *Journal of Algorithms* 21(2): 267–305.

Ramalingam, G., and Reps, T. 1996b. On the Computational Complexity of Dynamic Graph Problems. *Theoretical Computer Science* 158(1–2): 233–277.

Rohnert, H. 1985. A Dynamization of the All Pairs Least Cost Path Problem. In *Proceedings of the Second Annual Symposium on Theoretical Aspects of Computer Science,* 279–286. New York: Springer-Verlag.

Romero, L.; Morales, E.; and Sucar, E. 2001. An Exploration and Navigation Approach for Indoor Mobile Robots Considering Sensor's Perceptual Limitations. In Proceedings of the International Conference on Robotics and Automation, 3092–3097. Washington, D.C.: IEEE Computer Society.

Russell, S. 1992. Efficient Memory-Bounded Search Methods. In *Proceedings of the Tenth European Conference on Artificial Intelligence, ECAI92,* 1–5. Chichester, U.K.: Wiley.

Simmons, R. 1988. A Theory of Debugging Plans and Interpretations. In Proceedings of the Seventh National Conference on Artificial Intelligence, 94–99. Menlo Park, Calif.: American Association for Artificial Intelligence.

Spira, P., and Pan, A. 1975. On Finding and Updating Spanning Trees and Shortest Paths. *SIAM Journal on Computing* 4(3): 375–380.

Stentz, A. 1995. The Focused D* Algorithm for Real-Time Replanning. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1652–1659. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Stentz, A., and Hebert, M. 1995. A Complete Navigation System for Goal Acquisition in Unknown Environments. *Autonomous Robots* 2(2): 127–145.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction.* Cambridge, Mass.: MIT Press.

Tao, M.; Elssamadisy, A.; Flann, N.; and Abbott, B. 1997. Optimal Route Replanning for Mobile Robots: A Massively Parallel Incremental A* Algorithm. In Proceedings of the IEEE International Conference on Robotics and Automation, 2727–2732. Washington, D.C.: IEEE Computer Society.

Thayer, S.; Digney, B.; Diaz, M.; Stentz, A.; Nabbe, B.; and Hebert, M. 2000. Distributed Robotic Mapping of Extreme Environments. In Proceedings of the SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII, Volume 4195. Bellingham, WA: International Society for Optical Engineering.

Thrun, S. 1998. Lifelong Learning Algorithms. In *Learning to Learn,* eds. S. Thrun and L. Pratt, 181–209. New York: Kluwer Academic.

Thrun, S.; Bücken, A.; Burgard, W.; Fox, D.; Fröhlinghaus, T.; Hennig, D.; Hofmann, T.; Krell, M.; and Schmidt, T. 1998. Map Learning and High-Speed Navigation in RHINO. In *Artificial Intelligence– Based Mobile Robotics: Case Studies of Successful Robot Systems,* eds. D. Kortenkamp, R. Bonasso, and R. Murphy, 21–52. Cambridge, Mass.: MIT Press

Trovato, K. 1990. DIFFERENTIAL A*: An Adaptive Search Method Illustrated with Robot Path Planning for Moving Obstacles and Goals and an Uncertain Environment. *Journal of Pattern Recognition and Artificial Intelligence* 4(2): 245–268.

Veloso, M. 1994. *Planning and Learning by Analogical Reasoning*. New York: Springer.

Verfaillie, G., and Schiex, T. 1994. Solution Reuse in Dynamic Constraint-Satisfaction Problems. In Proceedings of the Twelfth National Conference on Artificial Intelligence, 307–312. Menlo Park, Calif.: American Association for Artificial Intelligence.

Watkins, C., and Dayan, P. 1992. Q-LEARNING. *Machine Learning* 8(3–4): 279–292.

Sven Koenig is an associate professor at the University of Southern California. His research interests are in the areas of decision-theoretic planning and robotics. He has coauthored over 70 publications in these areas and is the recipient of a National Science Foundation CAREER award and an IBM Faculty Partnership award. His e-mail address is skoenig@usc.edu.

Maxim Likhachev is a Ph.D. student at Carnegie Mellon University. His research interests are in robotics.

Yaxin Liu is a Ph.D. student at the Georgia Institute of Technology. His research interests are in the area of decision-theoretic planning. He is the recipient of an IBM graduate fellowship.

David Furcy is a Ph.D. student at the Georgia Institute of Technology. His research interests are in the area of search. His-email address is dfurcy@cc.gatech.edu.