A Comparison of Fast Search Methods for Real-Time Situated Agents

Sven Koenig*
Computer Science Department, University of Southern California
941 W 37th Street, Los Angeles, California 90089-0781
skoenig@usc.edu

Abstract

Real-time situated agents, including characters in real-time computer games, often do not know the terrain in advance but automatically observe it within a certain range around them. They have to interleave planning with movement to make planning tractable when moving autonomously to user-specified coordinates. Planning faces real-time requirements since it is important that the agents be responsive to the commands of the users and move smoothly. In this paper, we compare two fast search methods for this task that speed up planning in different ways, namely real-time heuristic search (LRTA) and incremental heuristic search (D* Lite), resulting in the first comparison of real-time and incremental heuristic search in the literature. We characterize when to choose which search method, depending on the kind of terrain and the planning objective.*

1. Introduction

Consider path planning for characters in real-time computer games such as Total Annihilation, Age of Empires or Dark Reign. These real-time situated agents often do not know the terrain in advance but automatically observe it within a certain range around them and then remember it for future use. To make the agents easy to control, one needs to give them the capability to understand and execute high-level user commands. For example, the users can click on certain coordinates in known or unknown terrain and the agents then move autonomously to these coordinates. These path-planning problems are interesting because they are different from the traditional off-line search problems encountered in fields other than autonomous agents. In particular, our agents might not know the terrain initially and the resulting large number of contingencies makes planning difficult. Finding optimal plans is often intractable since the agents would have to find large conditional plans to solve the planning tasks. However, planning faces real-time requirements since the agents need to be responsive to the commands of the users and move smoothly. Thus, they need to use planning techniques that make planning fast by sacrificing the optimality of the resulting plans. In this paper, we describe an agent architecture that interleaves planning with movement and then compare two fast search methods that can be used as part of this agent architecture. As the agents move in the terrain, they observe more of it, which speeds up planning during subsequent planning episodes since it reduces the number of possible contingencies. The resulting paths are likely not optimal but this is often outweighed by the computational savings gained. The first technique that we study is real-time heuristic search, which makes planning efficient by limiting the search horizon. The second technique is incremental heuristic search, which makes planning efficient by reusing information from previous planning episodes to speed up the current one. We compare a simple version of the real-time heuristic search method LRTA* [16] from artificial intelligence experimentally to the real-time heuristic search method D* Lite [14] from robotics, resulting in the first comparison of real-time and incremental heuristic search in the literature. We characterize when to choose which one of the two search methods, depending on the kind of terrain and the planning objective.

2. Planning Problem and Agent Architecture

We study fast search methods that move real-time situated agents autonomously to user-specified coordinates in initially unknown terrain. The terrain is discretized into cells that are either blocked or unblocked, a common practice in the context of real-time computer games [2]. We assume for simplicity that the agents can move in the four main compass directions with equal cost and thus operate on four-connected grids. As heuristic estimate of the distance of two cells we use the sum of the absolute differences of their x and y coordinates (Manhattan distance). The agents initially do not know which cells are blocked. They always know which (unblocked) cells they are in, sense the blockage status of their neighboring cells, and can then move to any one of the unblocked neighboring cells. Their task is to move to a given goal cell. To make planning tractable,

* This research has been partly supported by an NSF award to Sven Koenig under contract IIS-0350584. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the offi cial policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

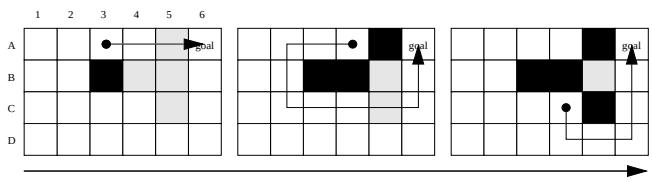

Figure 1. Obvious Planning Approach

we use an agent architecture that always assumes that cells whose blockage status has not been observed yet are unblocked (freespace assumption) [15]. If an action execution (movement) does not have the desired effect during plan execution because this assumption does not hold, the agents stop executing the plan and repeat the planning process from their new states, until the planning task is solved. Thus, they interleave planning with movement. They cannot inadvertently execute actions that make it impossible for them to reach the goal cell since our grids are undirected. The simplicity of our planning problem helps us to compare real-time and incremental heuristic search rigorously even those both search methods also apply to more complex scenarios, such as dynamically changing grids with more complex topologies and nonuniform edge costs as well as agents with larger sensor ranges. Thus, both of them can also be used in the presence of other agents.

3. Incremental Heuristic Search

One obvious approach to solving our planning problem is to always plan a complete path from the current cells of the agents to the goal cell. Since the agents always assume that all cells whose blockage status they have not observed yet are unblocked, they determine a shortest presumed unblocked path (a path that does not pass through cells that are known to be blocked) from their current cells to the goal cell and then move along it until they either reach the goal cell or observe that their current path is blocked. If they cannot find any presumed unblocked path, then there is no path from their start cell or, alternatively, their current cell to the goal cell. Figure 1 shows an example. The circle denotes the agent. Black cells denote blocked cells that the agent has already observed, and grey cells denote blocked cells of which the agent does not yet know. The arrows show the paths of the agent between replanning episodes.

Some theoretical properties of this planning approach have been studied in the literature. It is easy to show that the planning approach either moves agents to the goal cell or correctly reports that this is impossible. The resulting paths are reasonably short. In particular, the worst-case length of a path on graphs with n (blocked or unblocked) vertices is $O(n \log^2 n)$ on general graphs and $O(n \log n)$ on planar graphs, including grids [18]. The planning approach can be implemented efficiently with the incremental heuristic search methods focused Dynamic A* (D*) [22] and D* Lite [14], that both reuse information from previous planning episodes to speed up the current one. This is possible because the agents typically discover only a small number of blocked cells between planning episodes, and successive planning problems are thus very similar. D* is widely used in mobile robotics, including DARPA's Unmanned Ground Vehicle (UGV) program, Mars rover prototypes, and tactical mobile robot prototypes [9, 23]. We use D* Lite in our comparison since D* and D* Lite are about equally efficient but D* Lite is somewhat easier to understand.

4. Real-Time Heuristic Search

Another obvious approach to solving our planning problem is to plan only the beginning of a path from the current cells of the agents to the goal cell. This is sometimes called agent-centered search and can be done by restricting planning to the part of the terrain around the current cells of the agents (local search) [12]. The agents determine the local search space, search it, decide how to move within it, and move along this path until they either leave the local search space or observe that their current path is blocked. They repeat this process until they reach the goal cell. Real-time heuristic search is an agent-centered search technique that stores a value in memory for each state that it encounters during planning and uses techniques from asynchronous dynamic programming to update the values as planning progresses to make them more informed and, this way, avoid cycling forever. Learning Real-Time A* (LRTA*) [16] is probably the most popular real-time heuristic search method. The values of its states approximate the goal distances of the states. They can be initialized with a heuristic approximation of the goal distances to focus planning towards the goal cell.

Some theoretical properties of LRTA* have been studied in the literature. While LRTA* cycles forever if the goal cell cannot be reached, one can show that it moves the agents to the goal cell if it always updates at least the value of its current cell and the goal cell can be reached from every cell. The worst-case length of its paths on graphs with n unblocked vertices and diameter d is $O(nd)$ [13]. Real-time heuristic search methods have been used in artificial intelligence to solve large search tasks, including the twenty-four puzzle [17] and STRIPS-type planning tasks [4]. Simple versions of real-time heuristic search methods have also been studied in the context of situated agents [11, 7, 1].

Many versions of LRTA* have been suggested in the literature [11] since there are a large number of design choices, including how to determine the local search spaces and the values of which states in them to update. Since we compare LRTA* with a computationally intensive search method, we decided to design a version of LRTA* that is

Fast implementations of the following version of LRTA* do not initialize all values up front since many states might not get encountered during the search. Rather, they initialize a value only when it is needed for the first time. From then on, they retain the value in memory so that they can update it during the search.

1. Perform an A* search from the current state toward the goal state until either n states have been expanded or the goal state is about to be expanded. The expanded states form the local search space.
2. Use Dijkstra's algorithm to replace the value of each state in the local search space (that is, a state expanded in the first step) with the sum of the distance from the state to a state s that borders the local search space and the value of state s, minimized over all states s that border the local search space.
3. Follow the following path until the agents either leave the local search space or discover that the path is blocked, namely a shortest path from the current state to a state s that borders the local search space and minimizes the sum of the distance from the current state to state s and the value of state s, minimized over all states s that border the local search space. This path is followed automatically if the agents use the greedy strategy that repeatedly moves from the current state to the neighboring state s that minimizes the sum of the distance from the current state to state s and the (new) value of state s. (Ties should be broken so that the agents move along the edges of the A* search tree and thus stay in the local search space for as long as possible.)
4. If the current state is different from the goal state, then go to the first step, otherwise terminate successfully.

Figure 2. A Version of LRTA* (1)

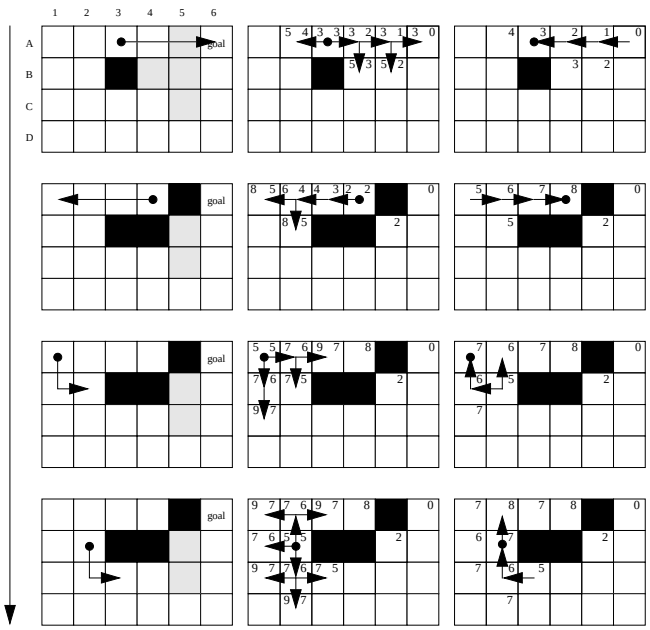

Figure 3. A Version of LRTA* (2)

also computationally intensive. Figure 2 describes the four steps of this version of LRTA* in more detail, which was inspired by an algorithm in [21].

It has not been studied extensively what the local search spaces of LRTA* should be. LRTA* is most often used with local search spaces that contain only the current states of the agents but their sizes should really be optimized for the planning objective. There have been some suggestions for choosing larger local search spaces [21, 20, 10] but they typically do not satisfy hard real-time constraints where only a certain amount of time is available for each planning episode. The easiest way of limiting the planning time is to limit the size of the local search spaces but this leaves open the question exactly which cells the local search spaces should contain. It makes sense that the local search spaces are continuous parts of the terrain that contain the current cells of the agents since these parts of the terrain contain the cells that the agents might soon be in and are thus immediately relevant for them in their current situations. In the following, we describe a simple way of choosing the local search spaces, namely by performing an A* search [19] from the current states of the agents toward the goal state until n states have been expanded, where n is an external parameter, called the lookahead. The idea behind using A* in the first step of our version of LRTA* is to try to reject the current path if additional planning time is available. The agent eventually moves towards a state s that borders the local search space and minimizes the sum of the distance from the current state to state s and the value of state s, minimized over all states s that border the local search space. This is a state that was generated but not expanded by the A* search and has the smallest f-value among all such states. Consequently, it is the state that the A* search would expand next if it were allowed to expand one additional state, which could change the current path. The local search space then consists of the expanded states. The idea behind using Dijkstra's algorithm in the second step of our version of LRTA* is to update the values of all states in the local search space to make them locally consistent [20] and thus propagate as much information as possible.

Figure 3 gives the beginning of an example of how our version of LRTA* operates. The lookahead is three. The left column shows how the agent moves, similar to Figure 1. The center column shows the results of the A* searches. Each cell with a cached value is labeled with it in the upper right corner. Each cell generated during an A* search is labeled with its f-value in the upper left corner. The arrows go from parents to their children in the search tree. The right column shows the results of Dijkstra's algorithm. Each cell with a cached value is labeled with it in the upper right corner. Each cell in the local search space has exactly one incoming arrow that shows the cell whose value was used to update its value. Note that only the beginning of the path of the agent is shown. The complete path is longer than the one from Figure 1.

Both A* and Dijkstra's algorithm expand each state at most once and are thus efficient. They have an interesting interface: Dijkstra's algorithm needs to initialize its prior-

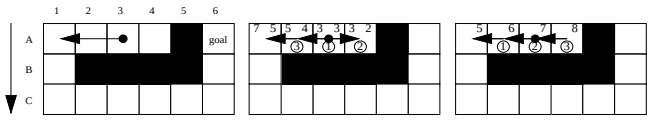

Figure 4. Interfacing A* and Dijkstra (1)

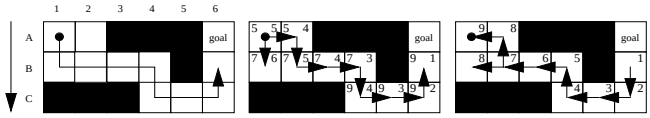

Figure 5. Interfacing A* and Dijkstra (2)

ity queue with the states that border the local search space but these are exactly the states is the priority queue of A* after A* terminates. Thus, Dijkstra's algorithm can reuse the priority queue of A*. However, it needs to change the priorities of the states in the priority queue. It appears that Dijkstra's algorithm should be able to reuse more information gathered during the A* search, such as the order in which A* expanded the states or the A* search tree. However, it is currently unclear how to exploit these ideas. Figure 4, for instance, gives an example of LRTA* with lookahead three in known terrain where the order in which Dijkstra's algorithm updates the states is different from both the order and the reverse of the order in which A* expanded the states. (The circled numbers show this order.) Similarly, Figure 5 gives an example of LRTA* with lookahead 9 in known terrain where the value of cell B1 gets calculated from the value of cell B2 even though these cells are not connected in the A* search tree.

5. Experimental Comparison

The ideas behind real-time and incremental heuristic search could, in principle, be combined by restricting planning to the part of the terrain around the current cells of the agents and reusing information from previous planning episodes to speed up the current one. However, current incremental heuristic search methods require that the root of the search tree does not change, although there has been some recent progress on relaxing this restriction [6]. Thus, incremental heuristic search is commonly used by searching from the goal cell towards the current cells of the agents, which does not work in conjunction with real-time heuristic search. The question then arises when to use real-time heuristic search and when to use incremental heuristic search. Incremental heuristic search has advantages over real-time heuristic search: Since incremental heuristic search plans a complete path, it can easily discover that the goal state cannot be reached from the current state. On the other hand, real-time heuristic search also has advantages over incremental heuristic search: Since real-time heuristic search plans only in the state space around the current cells of the agents, the sizes of the local search spaces can be freely determined, depending on how fast planning has to be. Thus, real-time heuristic search can satisfy hard real-time requirements in terrain of any size by choosing sufficiently small local search spaces (potentially at the cost of longer paths) while the planning time of incremental heuristic search increases with the size of the terrain. We now compare our version of LRTA* with the optimized final version of D* Lite as published in [14]. We use standard binary heaps to implement the priority queues of both search methods and test them on two different kinds of grids: one in which the heuristics can be very misleading and one in which they are generally not misleading. In each case, we average over 1000 (for mazes) and 5000 (for grids with random obstacles) randomly generated four-connected grids of size 301×301 with randomly chosen start and goal states with the restriction that there is a path from the start state to the goal state. In the following, we describe our results in detail. We measure all times in microseconds and all distances in number of movements.

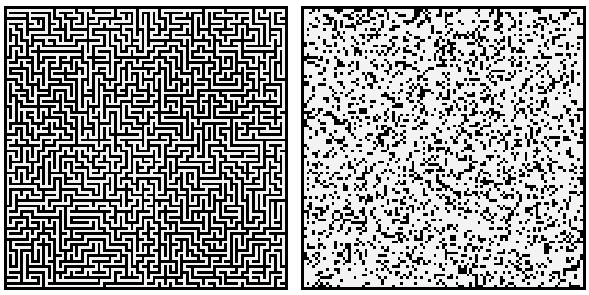

Figure 6. Maze (l) and Random Obstacles (r)

6. Mazes

We first use acyclic mazes as test domains whose corridor structure was generated with depth-first search. Figure 6 (left) shows an example (of smaller size than used in the experiments). As heuristic approximation of the distance of two states we consider the Manhattan distance, that is, the sum of the differences of their x- and y-coordinates (strong heuristics). We also consider the maximum of the differences of their x- and y-coordinates (weak heuristics). The strong heuristics dominate the weak heuristics, and thus A* with the strong heuristics expands fewer states than A* with the weak heuristics but finds paths of the same lengths. D* Lite is an incremental version of A*, and experimentally D* Lite with the strong heuristics also expands fewer states than D* with the weak heuristics but finds paths of about

(a) = state expansions per planning episode (lookahead), (b) = average planning time, and (c) = average number of movements (path length).

	mazes				grids with random obstacles			
	strong heuristics		weak heuristics		strong heuristics		weak heuristics	
(a)	(b)	(c)	(b)	(c)	(b)	(c)	(b)	(c)
	D* Lite							
-	357417.38	21737.53	373560.62	21140.40	36825.63	308.98	40737.34	313.78
	LRTA*							
1	985361.73	1987574.25	628175.97	1259958.00	28279.51	498.55	28292.81	363.16
3	640567.24	931230.38	477551.83	685570.04	28380.11	377.15	28446.67	363.30
5	441522.49	594675.73	366611.13	477525.03	28435.03	337.67	28568.88	339.47
7	395581.98	499083.52	321784.17	382949.65	28536.61	329.00	28658.00	327.50
9	358532.52	422475.56	296545.64	321547.69	28617.42	322.19	28769.02	318.39
11	313997.99	337704.16	277974.35	272841.73	28698.35	315.32	28877.77	315.32
13	302791.98	303562.40	276238.32	252374.66	28785.79	310.35	29008.12	315.57
15	290252.67	268652.32	281280.60	239072.72	28873.00	307.15	29118.05	314.14
17	284827.58	243952.42	277453.57	215615.93	28966.89	305.47	29226.43	311.46
19	276990.44	217852.00	280483.72	199517.41	29056.67	303.58	29341.93	311.65
21	279855.69	205370.41	273280.22	177142.96	29152.59	302.27	29476.83	311.11
23	285321.52	196601.90	283999.33	171600.71	29241.04	301.54	29585.83	310.20
25	274999.82	169685.19	283950.17	155736.31	29332.52	300.77	29701.80	309.44
27	293554.47	176642.26	292767.20	151277.34	29428.39	300.24	29822.47	309.43
29	293262.79	163418.55	296116.07	140895.33	29524.39	299.44	29949.31	309.84
...	...	...	...	...	...	...	...	...

Table 1. Results for Different Heuristics

the same lengths. However, Table 1 shows that this is not the case for LRTA*. LRTA* with the strong heuristics tends to expand more states than LRTA* with the weak heuristics but finds longer paths, at least for smaller lookaheads. The reason for this is that the relative differences of the heuristic values are much more important for focusing the search of LRTA* than their absolute values. Because the heuristics can be misleading in mazes, it is better if the differences of the heuristic values are small because LRTA* can then correct them faster. A similar phenomenon had previous been described for searching the eight-puzzle with LRTA* [13]. We use D* Lite and LRTA* with the heuristics that worked best for them. Thus, we use D* Lite with the strong heuristics and LRTA* with the weak heuristics.

Table 2 tabulates our results.[1] Comparing the planning time of D* Lite and LRTA* is difficult because the search spaces are relatively small. The reason for this is that maps of real-time computer games fit into memory and planning for each game character has to be fast, especially if the number of characters is large. Then, however, the scaling behavior of the search methods is less important than the hardware and implementation details – including the data structures, tie-breaking strategies, and coding tricks. To avoid this problem, one often uses proxies instead of the planning time itself, such as the number of state expansions. This is not possible in our case since D* Lite and LRTA* operate in very different ways, which forces us to compare their planning times directly. However, Figure 7 shows that the planning time of LRTA* with different lookaheads appears to be roughly proportional to its number of state expansions, which gives us hope that different hardware and implementation details change the planning time of LRTA* by only a constant factor.

1 Note that the number of state expansions can be smaller than the product of the lookahead and the number of planning episodes because, around the goal state, the number of state expansions per planning episode is smaller than the lookahead since state expansion stops once the goal state is about to be expanded.

(a) = state expansions per planning episode (lookahead) (b) = average number of state expansions, (c) = average number of planning episodes (d) = average number of movements (path length), (e) = average number of movements per planning episode, (f) = average planning time, (g) = average planning time per planning episode, and (h) = average planning time per movement.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
			D* Lite				
-	230893.54	6606.37	21737.53	3.29	357417.38	54.10	16.44
			LRTA*				
1	1259958.00	1259958.00	1259958.00	1.00	628175.97	0.50	0.50
3	1012633.01	337544.61	685570.04	2.03	477551.83	1.41	0.70
5	765644.80	153129.55	477525.03	3.12	366611.13	2.39	0.77
7	658618.41	94089.35	382949.65	4.07	321784.17	3.42	0.84
9	588810.14	65424.97	321547.69	4.91	296545.64	4.53	0.92
11	531955.23	48361.94	272841.73	5.64	277974.35	5.75	1.02
13	518431.33	39882.58	252374.66	6.33	276238.32	6.93	1.09
15	517913.09	34531.72	239072.72	6.92	281280.60	8.15	1.18
17	495466.48	29150.37	215615.93	7.40	277453.57	9.52	1.29
19	487622.82	25670.60	199517.41	7.77	280483.72	10.93	1.41
21	459565.74	21891.42	177142.96	8.09	273280.22	12.48	1.54
23	470419.04	20461.61	171600.71	8.39	283999.33	13.88	1.66
25	456751.93	18279.86	155736.31	8.52	283950.17	15.53	1.82
27	465707.78	17259.47	151277.34	8.76	292767.20	16.96	1.94
29	460964.20	15907.64	140895.33	8.86	296116.07	18.61	2.10
31	469144.66	15147.32	135554.16	8.95	310131.20	20.47	2.29
33	460947.12	13983.02	125789.79	9.00	304691.69	21.79	2.42
35	474447.93	13571.97	123304.98	9.09	315807.85	23.27	2.56
37	492481.84	13327.82	122274.35	9.17	329287.75	24.71	2.69
39	514415.51	13209.17	122839.82	9.30	344388.42	26.07	2.80
41	512638.46	12523.62	114917.08	9.18	348330.44	27.81	3.03
43	517674.55	12060.69	111242.82	9.22	354049.77	29.36	3.18
45	507532.55	11301.47	100257.67	8.87	354495.33	31.37	3.54
47	532693.30	11358.48	103038.69	9.07	370009.76	32.58	3.59
49	555105.16	11355.05	104059.85	9.16	385354.53	33.94	3.70

Table 2. Results for Mazes

Table 2 shows some interesting trends: First, the path length of LRTA* decreases as its lookahead increases: more planning results in shorter paths, which confirms earlier results in different domains [16] although exceptions to this property have also been reported [5]. Second, the planning time first decreases and then increases as the lookahead of LRTA* increases. This is the result of two different effects, namely an increasing planning time per planning episode due to the larger lookahead and a decreasing number of planning episodes due to the shorter paths. The graph in Figure 8 visualizes this trade-off between the planning time and the resulting path length. The planning time of LRTA* is larger than the one of D* Lite for lookaheads of more than 45, and its paths are longer than the ones of D* Lite for all tabulated lookaheads. Because the heuristics can be misleading in mazes, planning all the way to the goal cell allows one to find short paths. Thus, the paths of D* Lite are short. The path of LRTA* are short only for larger lookaheads. LRTA* with smaller lookaheads moves back and forth in local minima of the value surface until it has increased the values of the cells sufficiently to be able to escape the local minima, which results in longer paths.

In the following, we describe three planning objectives and analyze which search method one should choose for each one. We assume that one cannot overlap planning and movement and thus has to interleave them. This is not completely realistic especially if movement is slow but allows us to obtain first results. In future work, we intend to relax this assumption, which will require us to develop new theoretical foundations that might be similar to those in [3, 24, 8].

- We first study what to do if one wants to minimize

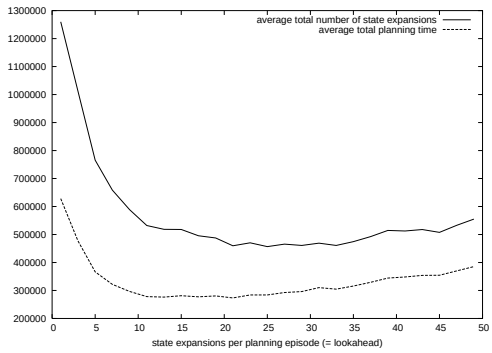

Figure 7. Planning Time vs. State Expansions

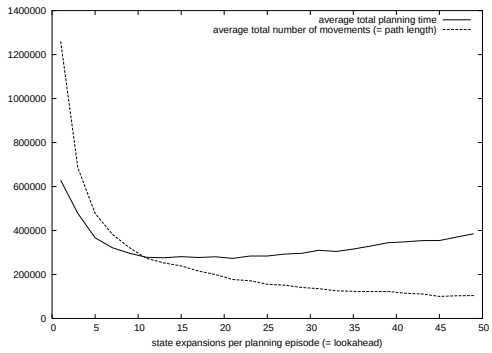

Figure 8. Planning Time vs. Path Length

the sum of the planning and plan-execution time. The result depends on the relative speed of planning and movement. If planning is sufficiently fast relative to movement (a realistic assumption for many applications), then the sum of the planning and plan-execution time is determined by the movement time, that is, path length (x is large, see below). In this case, a large lookahead is optimal for LRTA*. On the other hand, if movement is sufficiently fast relative to planning, then the sum of the planning and plan-execution time is determined by the planning time (x is small). In this case, a lookahead of 21 is optimal for LRTA*. To understand the cases between these extremes better, we use $x > 0$ to denote the ratio of the planning speed and movement speed. The sum of the planning and plan-execution time is then proportional to: planning time + $x \times$ path length. The following table shows the optimal lookahead for LRTA* as a function of x:

range of x	optimal lookahead
$10^{-4.00}$ -$10^{-0.31}$	21
$10^{-0.30}$ -$10^{-0.16}$	25
$10^{-0.15}$ -$10^{+0.29}$	33
...	...

This result is intuitive: Lookaheads that are smaller than the lookahead that minimizes the planning time cannot be optimal for LRTA* since both its planning time and its path length decrease as its lookahead increases. The lookahead that minimizes its planning time is optimal for $x = 0$. Its planning time increases and its path length decreases as the lookahead increases, starting with the lookahead that minimizes its planning time. Thus, its optimal lookahead increases as x increases. If x is larger than $10^{-0.27}$, then D* Lite should be preferred over LRTA* with the optimal lookahead since LRTA* needs a much larger planning time than D* Lite to find paths that are not much longer than the ones of D* Lite.

- We now study what to do if one wants to minimize the path length subject to the hard real-time constraint that only a certain amount of time is available for each planning episode. Thus, there is a time limit on the planning time per planning episode. We argue with averages in the following to make our argument simple even though this is not quite correct. D* Lite has a planning time per planning episode of 54.10 and thus cannot be used if the time limit is smaller than this. Thus, LRTA* with the largest lookahead that fits the time limit should be used instead of D* Lite if the time limit is smaller than 54.10. LRTA* with a lookahead of up to 75 (not shown in Table 2) has a planning time per planning episode that is smaller than 54.10. On the other hand, since LRTA* with a lookahead of even 241 (not shown in Table 2) finds only paths of length 44721.30 but has a planning time per planning episode of 168.19, D* Lite should be used if the time limit is larger than 54.10 (but small).

- Finally, we study what to do if one wants to minimize the path length subject to the hard real-time constraint that only a certain amount of time is available for each planning episode but the planning time can be amortized over the movements. Thus, there is a time limit on the planning time per movement. D* Lite has a planning time per movement of 16.44 and thus cannot be used if the time limit is smaller than this. Thus, LRTA* with the largest lookahead that fits the time limit should be used instead of D* Lite if the time limit is smaller than 16.44. LRTA* with a lookahead of up to 165 (not shown in Table 2) has a planning time per movement that is smaller than the one of D* Lite. On the other hand, since LRTA* with a lookahead of even 241 finds only paths of length 44721.30 but has a planning time per movement of 28.61, D* Lite should be used if the time limit is larger than 16.44 (but small).

The column headings (a) - (h) are the same as in Table 2.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
	D* Lite						
-	11424.90	72.54	308.98	4.26	36825.63	507.65	119.18
	LRTA*						
1	498.55	498.55	498.55	1.00	28279.51	56.72	56.72
3	622.46	207.83	377.15	1.81	28380.11	136.56	75.25
5	686.46	137.77	337.67	2.45	28435.03	206.39	84.21
7	796.09	114.30	329.00	2.88	28536.61	249.66	86.74
9	902.13	100.92	322.19	3.19	28617.42	283.58	88.82
11	1013.99	92.98	315.32	3.39	28698.35	308.65	91.01
13	1128.42	87.72	310.35	3.54	28785.79	328.14	92.75
15	1238.49	83.63	307.15	3.67	28873.00	345.25	94.00
17	1353.46	80.83	305.47	3.78	28966.89	358.36	94.83
19	1464.02	78.40	303.58	3.87	29056.67	370.61	95.71
21	1578.59	76.68	302.27	3.94	29152.59	380.16	96.45
23	1701.07	75.63	301.54	3.99	29241.04	386.62	96.97
25	1822.36	74.75	300.77	4.02	29332.52	392.40	97.53
27	1953.37	74.38	300.24	4.04	29428.39	395.67	98.02
29	2077.55	73.85	299.44	4.05	29524.39	399.76	98.60
31	2202.06	73.40	299.20	4.08	29615.10	403.48	98.98
33	2323.90	72.99	298.70	4.09	29715.34	407.11	99.48
35	2438.37	72.38	297.90	4.12	29799.11	411.73	100.03
37	2569.27	72.30	298.38	4.13	29904.07	413.58	100.22
39	2683.20	71.80	297.61	4.14	29994.63	417.73	100.78

Table 3. Results for Random Obstacles

(a) = state expansions per planning episode (lookahead), (b) = average number of state expansions (as an indicator for the average planning time - note, though, that breadth-first search can expand states faster than A*), and (c) = average number of movements (path length).

	mazes				grids with random obstacles			
	LRTA* (with A*)		LRTA* with BFS		LRTA* (with A*)		LRTA* with BFS	
(a)	(b)	(c)	(b)	(c)	(b)	(c)	(b)	(c)
1	1259958.00	1259958.00	1244573.34	1244573.34	498.55	498.55	496.82	496.82
3	1012633.01	685570.04	2151181.27	1427453.49	622.46	377.15	751.35	382.46
5	765644.80	477525.03	608563.97	339733.20	686.46	337.67	883.16	340.95
7	658618.41	382949.65	470885.88	239418.18	796.09	329.00	1081.67	331.05
9	588810.14	321547.69	439573.64	204921.68	902.13	322.19	1224.09	322.09
11	531955.23	272841.73	437526.96	189936.61	1013.99	315.32	1377.21	317.84
13	518431.33	252374.66	410861.79	165348.26	1128.42	310.35	1554.31	316.33
15	517913.09	239072.72	460207.28	177181.12	1238.49	307.15	1716.99	313.97
17	495466.48	215615.93	430183.56	154345.68	1353.46	305.47	1871.26	312.06
19	487622.82	199517.41	453565.17	154292.26	1464.02	303.58	2020.39	310.60
21	459565.74	177142.96	448383.49	144253.88	1578.59	302.27	2169.39	309.72
23	470419.04	171600.71	470230.40	144736.66	1701.07	301.54	2315.88	308.09
25	456751.93	155736.31	473433.00	138034.91	1822.36	300.77	2465.16	307.65
27	465707.78	151277.34	483322.87	135636.58	1953.37	300.24	2605.35	306.62
29	460964.20	140895.33	499253.40	133028.67	2077.55	299.44	2763.52	306.93
...	...	...	...	...	...	...	...	...

Table 4. Results for Different Search Spaces

7. Grids with Random Obstacles

We now use grids with randomly placed obstacles (blocked cells) as test domains. Their obstacle density is 25 percent. Figure 6 (right) shows an example (of smaller size than used in the experiments). Testing D* Lite and LRTA* on these different kinds of grids is interesting because they have very different properties from mazes. For example, the heuristics are not very misleading, and Table 1 shows that LRTA* performs about the same with the strong and the weak heuristics, at least for larger lookaheads. Thus, we now use both D* Lite and LRTA* with the strong heuristics. Table 3 tabulates our results, which are similar to the ones for mazes. However, the planning time per planning episode of both D* Lite and LRTA* is now much larger due to the larger branching factor. The planning time of LRTA* is now smaller than the one of D* Lite for all tabulated lookaheads (although eventually its planning time will be larger than the one of D* Lite as the lookahead increases since it does not use experience from previous planning episodes to speed up the current one), and its paths are shorter than the one of D* Lite for all tabulated lookaheads larger than 13. These good results for LRTA* are due to the fact that the heuristics are generally not misleading. One can therefore reduce the planning time by basically following the heuristics, with some lookahead to avoid being misled by inaccuracies of the heuristics caused by obstacles. This means that it is unnecessary to plan all the way to the goal cell. We do not understand yet why the path length of LRTA* with larger lookaheads is smaller than the one of D* Lite. We would have expected the path length of LRTA* to approach the one of D* Lite as the lookahead of LRTA* increases because LRTA* chooses paths of the same lengths as D* Lite once its local search spaces always border the goal state.

We now study again what to do if one wants to minimize the sum of the planning and plan-execution time. The following table shows the optimal lookahead for LRTA* as a function of x, the ratio of the planning speed and movement speed:

range of x	optimal lookahead
$10^{-4.00}$ -$10^{-0.09}$	1
$10^{-0.08}$ -$10^{+0.14}$	3
$10^{+0.15}$ -$10^{+1.06}$	5
$10^{+1.07}$ -$10^{+1.07}$	7
$10^{+1.08}$ -$10^{+1.24}$	11
$10^{+1.25}$ -$10^{+1.43}$	13
$10^{+1.44}$ -$10^{+1.71}$	15
$10^{+1.72}$ -$10^{+1.86}$	19
$10^{+1.87}$ -$10^{+2.07}$	21
$10^{+2.08}$ -$10^{+2.15}$	25
...	...

A lookahead of one now minimizes the planning time of LRTA*, which is smaller than the lookahead of 21 that minimized the planning time of LRTA* in mazes. For each value of x, LRTA* with the optimal lookahead has a smaller sum of the planning and plan-execution time than D* Lite, and thus LRTA* with the optimal lookahead should always be preferred over D* Lite. This is not surprising since both the planning time of LRTA* with a sufficiently large lookahead and the resulting path length are smaller than the ones of D* Lite. For the same reason, LRTA* should always be preferred over D* Lite if there is a time limit on the planning time per planning episode or movement.

8. Conclusions and Future Work

In this paper, we studied fast search methods that can be used for path planning by real-time situated agents such as characters in real-time computer games. We compared a simple version of LRTA*, a real-time heuristic search method, experimentally to D* Lite, an incremental heuristic search method, and characterized when to choose which one of the two search methods, depending on the kind of terrain and the planning objective. During our experiments,

we noticed that small algorithmic details can be very important. For example, it is beneficial for LRTA* with small lookaheads to generate the successors of states during the A* search in a random order rather than a fixed one. More generally, we do not yet understand what the most advantageous version of LRTA* is. For example, there are versions of LRTA* that reduce either the planning time or path length of our version of LRTA*. Some of them do not update all values in their local search space (for example, the original version of LRTA* [16]), update the values in a different way (for example, RTA* [16]), use faster search algorithms than A* to determine the local search spaces (for example, the original version of LRTA* [16]) or never expand cells with unknown blockage status (for example, LCM [20]). Thus, we need to compare different versions of LRTA* systematically. For example, the experimental results of this paper suggest that one should experiment with versions of LRTA* that are computationally less intensive than the version used in this paper. Table 4 shows first experimental results for a version of LRTA* with breadth-first search rather than A* search, which has a smaller planning time per planning episode because one can implement a breadth-first search without complex priority queues. The table shows that LRTA* with A* search performs well on grids with random obstacles but that LRTA* with breadth-first search performs well in mazes, at least for smaller lookaheads (different from three). In this case, LRTA* with A* search appears not to choose good local search spaces. We do not understand yet why this is the case. In the future, we intend to investigate additional real-time heuristic search methods and compare them with D* Lite and other ways of speeding up heuristic search, for example, using inconsistent heuristics rather than the two consistent heuristics used in this paper. A major open issue in this context is how to compare different search methods for situated agents since their planning time critically depends on both hardware and implementation details and the experimental results could therefore differ in different studies.

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 73(1):81–138, 1995.

[2] M. Bjornsson, M. Enzenberger, R. Holte, J. Schaeffer, and P. Yap. Comparison of different abstractions for pathfinding on maps. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2003.

[3] M. Boddy and T. Dean. Solving time-dependent planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 979–984, 1989.

[4] B. Bonet, G. Loerincs, and H. Geffner. A robust and fast action selection mechanism. In *Proceedings of the National Conference on Artificial Intelligence*, pages 714–719, 1997.

[5] V. Bulitko. Lookahead pathologies and meta-level control in real-time heuristic search. In *Proceedings of the Euromicro Conference on Real-Time Systems*, pages 13–16, 2003.

[6] S. Edelkamp. Updating shortest paths. In *Proceedings of the European Conference on Artificial Intelligence*, pages 655–659, 1998.

[7] M. Goldenberg, A. Kovarksy, X. Wu, and J. Schaeffer. Multiple agents moving target search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1538–1538, 2003.

[8] R. Goodwin. Reasoning about when to start acting. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 86–91, 1994.

[9] M. Hebert, R. McLachlan, and P. Chang. Experiments with driving modes for urban robots. In *Proceedings of the SPIE Mobile Robots*, 1999.

[10] T. Ishida. Moving target search with intelligence. In *Proceedings of the National Conference on Artificial Intelligence*, pages 525–532, 1992.

[11] T. Ishida. *Real-Time Search for Learning Autonomous Agents*. Kluwer Academic Publishers, 1997.

[12] S. Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 22(4):109–131, 2001.

[13] S. Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129:165–197, 2001.

[14] S. Koenig and M. Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, 2002.

[15] S. Koenig, C. Tovey, and Y. Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147:253–279, 2003.

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

[17] R. Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[18] A. Mugdal, C. Tovey, and S. Koenig. Analysis of greedy robot-navigation methods. In *Proceedings of the Conference on Artificial Intelligence and Mathematics*, 2004.

[19] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1985.

[20] J. Pemberton and R. Korf. Making locally optimal decisions on graphs with cycles. Technical Report 920004, Computer Science Department, University of California at Los Angeles, Los Angeles (California), 1992.

[21] S. Russell and E. Wefald. *Do the Right Thing – Studies in Limited Rationality*. MIT Press, 1991.

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

[23] S. Thayer, B. Digney, M. Diaz, A. Stentz, B. Nabbe, and M. Hebert. Distributed robotic mapping of extreme environments. In *Proceedings of the SPIE: Mobile Robots XV and Telemanipulator and Telepresence Technologies VII*, volume 4195, 2000.

[24] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, Computer Science Department, University of California at Berkeley, Berkeley (California), 1993.