Real-Time Heuristic Search: Research Issues

Sven Koenig
Georgia Institute of Technology
College of Computing
Atlanta, GA 30305-0280
skoenig@cc.gatech.edu

Abstract

Real-time (heuristic) search methods allow for fine-grained control over how much planning to do between plan executions. Many real-time search methods can use heuristic knowledge to guide planning, be interrupted at any state and resume execution at a different state, and improve their plan-execution time as they solve similar planning tasks, until their plan-execution time is optimal. Unfortunately, the behavior of real-time search methods is not yet well understood. In this paper, we show that the behavior of real-time search methods often differs from the behavior of more traditional and well-studied search methods and argue that it is important to investigate how properties of the heuristic functions, the domains, and the real-time search methods themselves influence their performance.

Introduction

Situated agents are agents that have to act in the world to achieve their goals. A central problem for them is how to achieve given goal states. Traditional search methods from artificial intelligence, such as A* (Nilsson 1971), first plan and then execute the resulting plan. The sum of planning and plan execution time, however, can often be reduced by interleaving planning and plan execution. In this paper, we discuss real-time (heuristic) search methods (Korf 1990), because these domain-independent search methods provide an efficient way for interleaving planning and plan execution. Real-time search methods are based on agent-centered search, like many heuristic SAT-solution methods. Agent-centered search methods restrict the search to a small part of the domain that can be reached from the current state with a small number of action executions. This is the part of the domain that is immediately relevant in the current situation. Many real-time search methods have the following advantageous properties: First, they allow for fine-grained control over how much planning to do between plan executions and thus are any-time contract methods (Russell and Zilberstein 1991). Second, they can use heuristic knowledge to guide planning, which reduces planning time without sacrificing plan-execution time. Third, they can be interrupted at any state and resume execution at a different state. Fourth, they can amortize learning over several search episodes, which allows them to find a plan with a suboptimal plan-execution time fast and then improve the plan-execution time as they solve similar planning tasks, until the plan-execution time is optimal. Thus, they can always have a small sum of planning and plan-execution time and still minimize the plan-execution time in the long run in case similar planning tasks unexpectedly repeat. This is important because no search method that interleaves planning and plan execution can guarantee to minimize the plan-execution time right away. Real-time search methods have been shown to be efficient alternatives to more traditional search methods. For example, they are among the few search methods that are able to find plans for the twenty-four puzzle, a sliding-tile puzzle with more than 7×10^{24} states (Korf 1993). They have also been used to solve large STRIPS-type planning tasks (Bonet *et al.* 1997) and large POMDP-type planning tasks (Geffner and Bonet 1998). Finally, studying real-time search methods can contribute to a better understanding of goal-directed reinforcement-learning methods since both classes of methods are similar (Koenig and Simmons 1993; Barto *et al.* 1995) and most reinforcement-learning researchers have not yet analyzed the plan-execution time of their methods (Koenig and Simmons 1996b). Unfortunately, different from more traditional search methods, that have been studied extensively, not much is known about real-time search methods. In this paper, we describe some promising areas for future research, and illustrate them with examples that we discovered as byproducts of our own research on real-time search methods (Koenig 1997).

Learning Real-Time A*

In this section, we describe one particular real-time search method, namely Korf's Learning Real-Time A* (LRTA*) method (Korf 1990), which is probably the most popular real-time search method. We use the following notation: S denotes the finite set of states of the domain, $s_{start} \in S$ the start state, and $G \subseteq S$ the set of goal states. The number of states is $n := |S|$. $A(s) \neq \emptyset$ is the finite, nonempty set of actions that can be executed in state $s \in S$. $succ(s, a)$ denotes the successor state that results from the execution of action $a \in A(s)$ in state $s \in S$. $gd(s)$ denotes the goal distance of state $s \in S$ (here: measured in action executions), and d is the largest goal distance of any state. We also use two operators with the following semantics: The

Initially, the u-values $u(s)$ are initialized with $f(s)$ for all $s \in S$, where f is a heuristic function for the goal distance of the states.

1. s := s_{start}.
2. If $s \in G$, then stop successfully.
3. a := one-of arg $\min_{a \in A(s)} u(succ(s,a))$.
4. $u(s)$:= $\max(u(s), 1 + u(succ(s,a)))$.
5. Execute action a.
6. s := $succ(s,a)$.
7. Go to 2.

Figure 1: LRTA*

expression "arg $\min_{x \in X} f(x)$" returns the elements $x \in X$ that minimize $f(x)$. Given such a set Y, the expression "one-of Y" returns an element of Y according to an arbitrary rule. Using this notation, we describe a simple version of LRTA* (with lookahead one), see Figure 1. It associates a small amount of information with the states that allows it to remember where it has searched already. In particular, it associates a u-value $u(s)$ with each state $s \in S$. The u-values approximate the goal distances of the states. LRTA* updates them as the search progresses and uses them to determine which actions to execute. In particular, LRTA* consists of a termination-checking step (line 2), an action-selection step (line 3), a value-update step (line 4), and an action-execution step (line 5). First, LRTA* checks whether it has reached a goal state and thus can terminate successfully. If not, it decides on the action to execute next. It looks only one action execution ahead and greedily picks the action that leads to a successor state with a smallest u-value. It then replaces the u-value $u(s)$ of the current state with the one-step lookahead value $\max(u(s), 1+u(succ(s,a)))$, which is a more accurate estimate. Finally, it executes the selected action and iterates.

Performance of LRTA*

In this section, we give an upper bound on the plan-execution time of LRTA* (here: measured in action executions) until it reaches a goal state (its "performance"). Bounding the plan-execution time also bounds the sum of planning and plan-execution time: For sufficiently slowly moving agents, the sum of planning and plan-execution time is almost completely determined by the plan-execution time. For sufficiently fast moving agents, on the other hand, the sum of planning and plan execution time is almost completely determined by the planning time, which is roughly proportional to the number of action executions (and thus the plan-execution time) since LRTA* performs only a constant amount of computation between action executions if all states have roughly the same (constant) number of successor states.

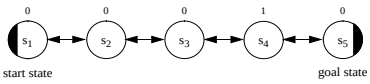

Figure 2: One-Dimensional Gridworld

Eight Puzzle with the "American" Goal State

1	2	3
8		4
7	6	5

Eight Puzzle with the "European" Goal State

1	2	3
4	5	6
7	8	

Figure 3: Eight Puzzle with Two Possible Goal States

Theorem 1 *LRTA* with an admissible heuristic function f has a plan-execution time of at most*

$$2\sum_{s \in S}\big[gd(s) - f(s)\big] + f(s_{start}) \quad \leq \quad 2\sum_{s \in S} gd(s)$$

action executions.

The proof of Theorem 1 can be found in (Koenig and Simmons 1995). The theorem implies that LRTA* with an admissible heuristic function has a plan-execution time of $O(nd)$ action executions. If one can reach a goal state from every state, then $d \leq n-1$, and LRTA* with an admissible heuristic function has a plan-execution time of $O(n^2)$ action executions. Thus, its plan-execution time is finite, which proves its correctness since LRTA* stops only in goal states. It can be shown that the bound provided by Theorem 1 is tight for both LRTA* with a completely informed heuristic function (that is, $f(s) = gd(s)$ for all $s \in S$) and LRTA* with a completely uninformed heuristic function (that is, $f(s) = 0$ for all $s \in S$) (Koenig and Simmons 1995). However, the bound is, in general, not representative for the average plan-execution time of LRTA* for a given planning task. In the following, we show that the behavior of LRTA* often differs from the behavior of more traditional and well-studied search methods, such as A*. This raises the question of how properties of the heuristic functions, the domains, and the real-time search methods themselves influence their plan-execution times.

Properties of Heuristic Functions

In this section, we show that properties of heuristic functions influence the behavior of LRTA* and A* differently although both search methods utilize similar properties of heuristic functions, such as their admissibility or consistency. We say that a heuristic function dominates another one if, for every state, the heuristic value that it assigns to that state is at least as large as the heuristic value that the other heuristic functions assigns to the state. We know that one can never degrade the performance of A* by switching from a consistent heuristic function to another consistent heuristic function that dominates the other one (Pearl 1985). However, this is not the case for LRTA* even though the upper bound on its plan-execution time is guaranteed to decrease. As an example, consider the one-dimensional

grid-world from Figure 2. LRTA* with a completely uninformed heuristic function always moves to the right and reaches the goal state on a shortest path from the start state. Now assume that LRTA* uses the consistent heuristic function shown above the states, that dominates the completely uninformed heuristic function. In this case, LRTA* does not reach the goal state any longer on a shortest path from the start state if ties are broken in favor of successor states with smaller indices. In this case, it traverses the state sequence s_1, s_2, s_3, s_2, s_1, s_2, s_3, s_4, and s_5. The same effect can also be observed in more complex domains (Koenig 1995). As an example, consider the following two heuristic functions for the eight puzzle with the "American" goal state, see Figure 3 (Pearl 1985). *Gaschnig's heuristic:* the smallest number of moves needed to achieve the goal state if it counts as one move when a tile is removed from its current square and placed on the empty square; and *the Tiles-Out-Of-Order heuristic:* the smallest number of moves needed to achieve the goal state if two or more tiles can occupy the same square and it counts as one move when a tile is removed from its current square and placed on any other square. Gaschnig's heuristic dominates the Tiles-Out-Of-Order heuristic, and both heuristic functions are consistent. Thus, the performance of A* with Gaschnig's heuristic is at least as good as the performance of A* with the Tiles-Out-Of-Order heuristic. The opposite is true for LRTA*. We average its plan-execution time over 25,000 examples and break ties among actions randomly. LRTA* with the Tiles-Out-Of-Order heuristic has, on average, a plan-execution time of 1,410 action executions, compared to 2,236 action executions for LRTA* with Gaschnig's heuristic. Out of the 25,000 runs, LRTA* with the Tiles-Out-Of-Order heuristic outperforms LRTA* with Gaschnig's heuristic 15287 times, is beaten 9682 times, and ties 31 times. These examples show that the performance of LRTA* is not completely correlated with the informedness of the heuristic functions even if they are consistent. This is due to local minima in the "value surface" of the heuristic function (Ishida 1997). Since LRTA* always chooses the action for execution that leads to a successor state with a smallest u-value, we expect LRTA* to do well if there is a good chance that it comes across a goal state when it mostly performs steepest decent on the initial u-values. This means that the differences in u-values of the successor states are more important than how close the u-values are to the goal distances. Consequently, the fewer local minima there are in the initial u-value surface and the "shallower" they are, the better we expect the plan-execution time of LRTA* to be. In general, however, not much is known about how heuristic knowledge affects the plan-execution time of real-time search methods, and there are no good techniques yet for predicting how well they perform with given heuristic functions. This is a promising area for future research.

Properties of Domains

In this section, we show that domain properties influence the behavior of LRTA* and A* differently. A* with a consistent heuristic function, for example, never expands states

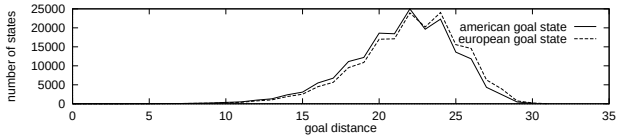

Figure 4: Goal Distances of the Eight Puzzle

that are further away from the start state than the closest goal state. LRTA* with the same heuristic function, on the other hand, can visit these states. According to Theorem 1, the plan-execution time of LRTA* depends on the goal distances of all states (and thus the average goal distance over all states), not only the goal distance of the start state. As an example, consider sliding-tile puzzles, which are sometimes considered to be hard planning domains because they have a small goal density. The eight puzzle, for example, has 181,440 states, but only one goal state. Although increasing the goal density tends to decrease the average goal distance, there exist planning domains with small average goal distance *and* small goal density (Koenig and Simmons 1995). The eight puzzle is an example: Figure 4 shows for every goal distance how many of the 181,440 states have this particular goal distance. The average goal distance of the eight puzzle with the "American" goal state is only 21.5 and its maximal goal distance is 30. Similarly, the average goal distance of the eight puzzle with the "European" goal state is 22.0 and its maximal goal distance is 31. More extensive statistics on the eight puzzle can be found in (Reinefeld 1993). Although the average goal distance could be as large as $(n-1)/2 = 90,719.5$, it is much smaller for the eight puzzle with either goal state. This means that LRTA* can never move far away from the goal states even if it executes suboptimal actions. Thus, the eight puzzle is not particularly hard to search with LRTA* among all domains with the same number of states.

Another property that makes domains easy to search with some real-time search methods, but not A*, is being Eulerian (Koenig and Simmons 1996a). Eulerian domains are directed domains where every state has as many actions that enter it as actions that leave it and, thus, Eulerian domains are a superset of undirected (more precisely: "bi-directed") domains. In general, however, not much is known about how domain properties affect the plan-execution time of real-time search methods, and there are no good techniques yet for predicting how well they will perform in a given domain. This is a promising area for future research.

Properties of Search Methods

In this section, we show that the plan-execution times of two similar real-time search methods can be very different, although experimental results in standard test domains indicate that both methods perform equally well. As an example, consider Node Counting, a real-time search method that differs from LRTA* only in line 4, see Figure 5. Variants of Node Counting have been used in (Pirzadeh and Snyder 1990; Thrun 1992; Balch and Arkin 1993), among

1. $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. $a :=$ one-of $\arg\min_{a \in A(s)} u(succ(s,a))$.
4. $u(s) := 1 + u(s)$.
5. Execute action a.
6. $s := succ(s,a)$.
7. Go to 2.

Figure 5: Node Counting

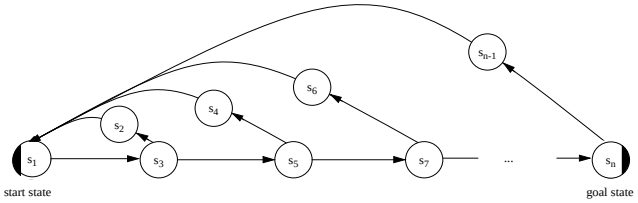

Figure 6: Reset State Space

others. The difference between Node Counting and LRTA* is the following: The u-values of (uninformed) LRTA* approximate the goal distances. LRTA* always moves to a successor state with a smallest u-value because it wants to get to states with smaller goal distances to eventually reach a state with goal distance zero, that is, a goal state. The u-values of uninformed Node Counting, on the other hand, correspond to the number of times Node Counting has visited the states. Node Counting always moves to a successor state with a smallest u-value because it wants to get to states which it has visited less frequently to eventually reach a state that it has not yet visited at all, that is, a possible goal state. Node Counting and LRTA* perform equally well in many domains. Consider, for example, the following two experiments in which the difference in plan-execution times of the two methods is statistically insignificant for any reasonable level of significance, for both sign tests and t tests (Koenig and Simmons 1996a). In the first experiment, we compare uninformed Node Counting and uninformed LRTA* on the eight puzzle with the "American" goal state. We average their plan-execution times over 25,000 examples and break ties among actions randomly. Node Counting has, on average, a plan-execution time of 85,579 action executions, compared to 85,746 action executions for LRTA*. Out of the 25,000 runs, Node Counting outperforms LRTA* 12,512 times and is beaten 12,488 times. In the second experiment, we use an obstacle-free grid-world of size 50×50 whose start and goal states are in opposite corners. Node Counting has, on average, a plan-execution time of 2,874 action executions, compared to 2,830 action executions for LRTA*. Out of the 25,000 runs, Node Counting outperforms LRTA* 12,345 times, is beaten 12,621 times, and ties 34 times. These two experiments seems to suggest that Node Counting and LRTA* perform equally well. A formal analysis, however, comes to a different conclusion. If one can reach a goal state from every state, then Theorem 1 implies that the plan-execution time of uninformed LRTA* is at worst quadratic in the number of states. The plan-execution time of uninformed Node Counting, however, can be exponential in the number of states. As an example, consider a variant of the "reset state spaces" used in (Koenig and Simmons 1996a). A reset state space is a domain in which all states (but the start state) have an action that leads back to the start state (here: via an intermediate state), see Figure 6. We say that the action "resets" Node Counting to the start state. Uninformed Node Counting has a plan-execution time of $2^{(n+1)/2} - 3$ action executions (for odd $n \geq 3$) if ties are broken in favor of successor states with smaller indices. For example, for $n = 7$, it traverses the state sequence s_1, s_3, s_2, s_1, s_3, s_5, s_4, s_1, s_3, s_2, s_1, s_3, s_5, and s_7. A more complex example can be used to show that the plan-execution time of uninformed Node Counting can be exponential in the number of states even in undirected domains (Koenig and Szymanski 1998). This result shows that sliding-tile puzzles and grid-worlds are not able to distinguish between real-time search methods that are always efficient, such as LRTA*, and real-time search methods that can be intractable, such as Node Counting. In general, however, not much is known about how different real-time search methods compare, and there are no good techniques yet for predicting which one will outperform the others on a given planning task. This is a promising area for future research.

Conclusions

Our results imply that real-time search methods have different properties than more traditional and well-studied search methods, such as A*. This motivates our argument that it is important to investigate how properties of the heuristic functions, the domains, and the real-time search methods themselves influence their plan-execution times. These results can help experimental researchers to distinguish easy planning tasks for real-time search methods from hard ones, which can help them to decide whether to use real-time search methods to solve a given planning task. If the answer is positive, the results can also help them to select an appropriate real-time search method. Finally, the results can help them to choose appropriate test-beds for experimenting with real-time search methods, reporting their results, and interpreting the results reported by others (Koenig and Simmons 1996a).

Another promising research direction, not mentioned earlier in this paper, is the application of real-time search methods to nondeterministic planning tasks (Koenig and Simmons 1995; Barto *et al.* 1995). In nondeterministic domains, real-time search methods have an additional advantage over search methods that first plan and only then execute the resulting plan, namely that they allow agents to gather information early. This information can be used to resolve some of the uncertainty and reduce the amount of planning done for unencountered situations, which makes planning more efficient. Real-time search methods have already been applied to some nondeterministic planning tasks, including robot navigation tasks (Koenig and Simmons 1998), but not much is known yet about their behavior in nondeterministic

domains. This is another promising area for future research.

References

Balch, T. and Arkin, R. 1993. Avoiding the past: A simple, but effective strategy for reactive navigation. In *International Conference on Robotics and Automation*. 678–685.

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

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism. In *Proceedings of the National Conference on Artificial Intelligence*.

Geffner, H. and Bonet, B. 1998. Solving large POMDPs by real-time dynamic programming. Technical report, Departamento de Computación, Universidad Simón Bolivar, Caracas (Venezuela).

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

Koenig, S. and Simmons, R.G. 1992. Complexity analysis of real-time reinforcement learning applied to finding shortest paths in deterministic domains. Technical Report CMU–CS–93–106, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Koenig, S. and Simmons, R.G. 1993. Complexity analysis of real-time reinforcement learning. In *Proceedings of the National Conference on Artificial Intelligence*. 99–105.

Koenig, S. and Simmons, R.G. 1995. Real-time search in non-deterministic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1660–1667.

Koenig, S. and Simmons, R.G. 1996a. Easy and hard testbeds for real-time search algorithms. In *Proceedings of the National Conference on Artificial Intelligence*. 279–285.

Koenig, S. and Simmons, R.G. 1996b. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning* 22(1/3):227–250.

Koenig, S. and Simmons, R.G. 1998. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*.

Koenig, S. and Szymanski, B. 1998. Node Counting and LRTA* for control: Are they equally good? (submitted).

Koenig, S. 1992. The complexity of real-time search. Technical Report CMU–CS–92–145, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Koenig, S. 1995. Agent-centered search: Situated search with small look-ahead. Phd thesis proposal, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Koenig, S. 1997. *Goal-Directed Acting with Incomplete Information*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

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

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

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

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

Pirzadeh, A. and Snyder, W. 1990. A unified solution to coverage and search in explored and unexplored terrains using indirect control. In *Proceedings of the International Conference on Robotics and Automation*. 2113–2119.

Reinefeld, A. 1993. Complete solution of the eight-puzzle and the benefit of node ordering in IDA*. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 248–253.

Russell, S. and Zilberstein, S. 1991. Composing real-time systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 212–217.

Thrun, S. 1992. The role of exploration in learning control with neural networks. In White, D. and Sofge, D., editors 1992, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. Van Nostrand Reinhold. 527–559.