Agent-Centered Search

Sven Koenig
College of Computing
Georgia Institute of Technology
skoenig@cc.gatech.edu

Abstract

In this article, we describe agent-centered search (sometimes also called real-time search or local search) and illustrate this planning paradigm with examples. Agent-centered search methods interleave planning and plan execution and restrict planning to the part of the domain around the current state of the agent, for example, the current location of a mobile robot or the current board position of a game. They can execute actions in the presence of time constraints and often have a small sum of planning and execution cost, both because they trade-off planning and execution cost and because they allow agents to gather information early in nondeterministic domains, which reduces the amount of planning they have to perform for unencountered situations. These advantages become important as more intelligent systems are interfaced with the world and have to operate autonomously in complex environments. Agent-centered search methods have been applied to a variety of domains, including traditional search, STRIPS-type planning, moving-target search, planning with totally and partially observable Markov decision processes models, reinforcement learning, constraint satisfaction, and robot navigation. We discuss the design and properties of several agent-centered search methods, focusing on robot exploration and localization.

Overview

Artificial intelligence has studied in detail off-line planning methods that first determine sequential or conditional plans (including reactive plans) and then execute them in the world. However, interleaving or overlapping planning and plan execution often has advantages for intelligent systems ("agents") that interact directly with the world. In this article, we study a particular class of planning methods that interleave planning and plan execution, namely agent-centered search methods (Koenig 1996; Koenig 1997a). Agent-centered search methods restrict planning to the part of the domain around the current state of the agent, for example, the current location of a mobile robot or the current board position of a game. The part of the domain around the current state of the agent is the part of the domain that is immediately relevant for the agent in its current situation (because it contains the states that the agent will soon be in) and sometimes might be the only part of the domain that the agent knows about. Figure 1 illustrates this approach. Agent-centered search methods usually do not plan all the way from the start state to a goal state. Instead, they decide on the local search space, search it, and determine which actions to execute within it. Then, they execute these actions (or only the first action) and repeat the overall process from their new state, until they reach a goal state. They are special kinds of any-time algorithms and share their advantages. By keeping the planning cost (here: time) between plan executions small, agent-centered search methods allow agents to execute actions in the presence of time constraints. By adapting the planning cost between plan executions to the planning and execution speeds of agents, agent-centered search methods allow agents to reduce the sum of planning and execution cost.

Agent-centered search is not yet a common term in artificial intelligence, although planning methods that fit its definition are scattered throughout the literature on artificial intelligence and robotics. In this article, we illustrate the concept of agent-centered search in deterministic and nondeterministic domains, describe which kinds of planning tasks they are suitable for, and give an overview of some agent-centered search methods from the literature that solve real-world planning tasks as part of complete agent architectures. We illustrate agent-centered search in nondeterministic domains using robot-navigation tasks such as repeated terrain coverage, exploration (map building), and localization. These tasks were performed by robots that have been used in programming classes, entered robot competitions, guided tours in museums, and explored natural outdoor terrain. By showing that different planning methods fit the same planning paradigm, we hope to establish a unified view that helps focus research on what we consider to be an exciting area of artificial intelligence.

Overview of Agent-Centered Search

The best known example of agent-centered search is probably *game playing*, such as playing chess. In this case, the states correspond to board positions and the current state corresponds to the current board position. Game-playing programs typically perform a minimax search with a lim-

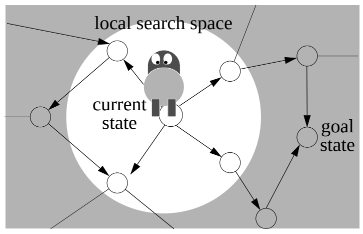


Figure 1: Agent-Centered Search

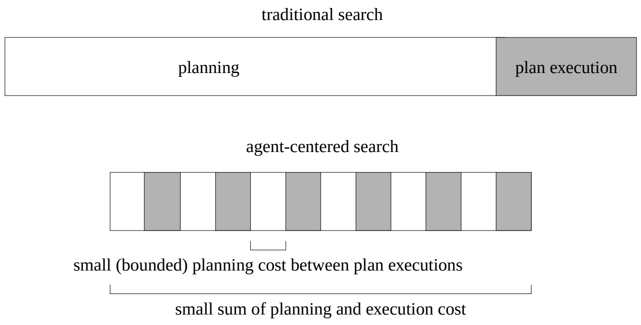

traditional search
planning
plan execution
agent-centered search
small (bounded) planning cost between plan executions
small sum of planning and execution cost

Figure 2: Traditional Search versus Agent-Centered Search

ited lookahead depth around the current board position to determine which move to perform next. Thus, they perform agent-centered search even though they are free to explore any part of the state space. The reason for performing only a limited local search is that the state spaces of realistic games are too large to perform complete searches in a reasonable amount of time. The future moves of the opponent cannot be predicted with certainty, which makes the planning tasks nondeterministic. This results in an information limitation that can only be overcome by enumerating all possible moves of the opponent, which results in large search spaces. Performing agent-centered search allows game-playing programs to choose a move in a reasonable amount of time while focusing on the part of the state space that is the most relevant to the next move decision.

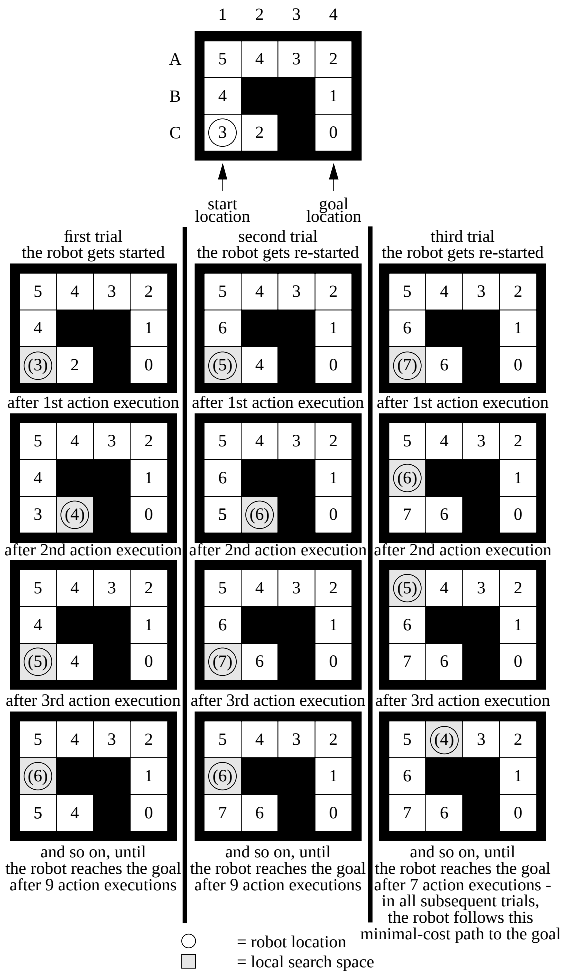
In this article, we concentrate on agent-centered search in single-agent domains. Traditional search methods, such as A* (Nilsson 1971; Pearl 1985), first determine plans with minimal execution cost (such as time or power consumption) and then execute them. Thus, they are off-line planning methods. Agent-centered search methods, on the other hand, interleave planning and execution and are thus on-line planning methods. They can have the following two advantages, as shown in Figure 2: They can execute actions in the presence of time constraints and often decrease the sum of planning and execution cost.

- **Time constraints:** Agent-centered search methods can execute actions in the presence of soft or hard time constraints. The planning objective in this case is to approximately minimize the execution cost subject to the constraint that the planning cost (here: time) between action executions is bounded. This objective was the original intent behind developing real-time (heuristic) search (Korf 1990) and includes situations where it is more important to act reasonably in a timely manner than to minimize the execution cost after a long delay. *Driving, balancing poles, and juggling devil sticks* are examples. For instance, before an automated car has determined how to negotiate a curve with minimal execution cost, it has likely crashed already. Another example is real-time *simulation and animation*, which become increasingly important for training and entertainment purposes, including real-time computer games. It is not convincing if an animated character sits there motionlessly until a minimal-cost plan has been found and then executes the plan quickly. Rather, it has to avoid artificial idle times and move smoothly. This objective can be achieved by keeping the amount of planning between plan executions small and approximately constant.

- **Sum of planning and execution cost:** Agent-centered search methods often decrease the sum of planning and execution cost compared to planning methods that first determine plans with minimal execution cost and then execute them. This property is important for planning tasks that need to be solved only once. The planning objective in this case is to approximately minimize the sum of planning and execution cost. *Delivery* is an example. If I ask my delivery robot to fetch me a cup of coffee, then I do not mind if the robot sits there motionlessly and plans for a while, but I do care about receiving my coffee as quickly as possible, that is, with a small sum of planning and execution cost. Since agents that perform agent-centered search execute actions before they know that the actions minimize the execution cost, they are likely to incur some overhead in execution cost but this increase in execution cost is often outweighed by a reduction in planning cost, especially since determining plans with minimal execution cost is often intractable, such as for the localization problems discussed in this article. How much (and where) to plan can be determined automatically (even dynamically), using either techniques tailored to specific agent-centered search methods (Ishida 1992) or general techniques from limited rationality and deliberation scheduling (Boddy and Dean 1989; Horvitz *et al.* 1989; Zilberstein 1993). Applications of these techniques to agent-centered search are described in (Russell and Wefald 1991).

To make our discussion more concrete, we now describe an example of an agent-centered search method in single-agent domains. We will relate all of the following agent-centered search methods to this one.

Learning Real-Time A* (LRTA*) (Korf 1990) is an agent-centered search method that stores a value in memory for each state that it encounters during planning and uses techniques from asynchronous dynamic programming (Bertsekas and Tsitsiklis 1997) to update the state values as plan-



1 2 3 4
A B C
start location
goal location
first trial the robot gets started
second trial the robot gets re-started
third trial the robot gets re-started
after 1st action execution
after 2nd action execution
after 3rd action execution
and so on, until the robot reaches the goal after 9 action executions
and so on, until the robot reaches the goal after 7 action executions - in all subsequent trials, the robot follows this minimal-cost path to the goal
= robot location
= local search space

Figure 3: LRTA* in a Simple Grid World

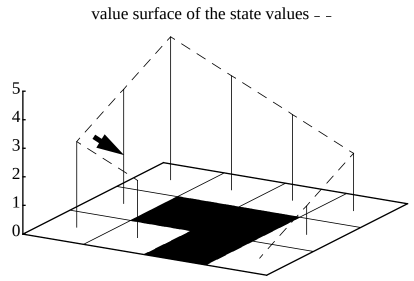
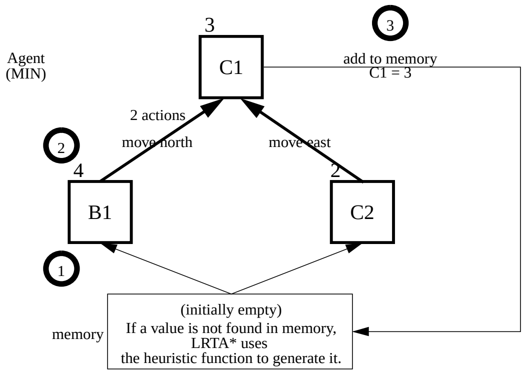


Figure 4: Initial Value Surface



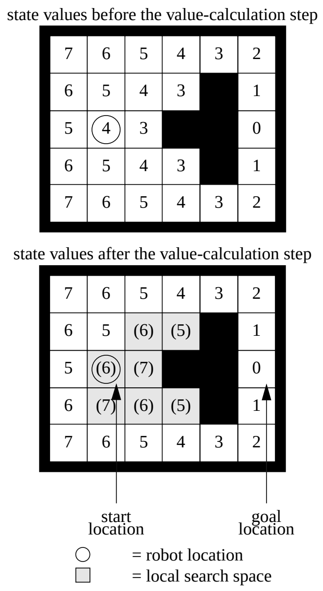
Agent (MIN)
3
C1
add to memory C1 = 3
2 actions
move north
move east
4
2
B1
C2
memory
(initially empty) If a value is not found in memory, LRTA* uses the heuristic function to generate it.

Figure 5: LRTA*

ning progresses. We refer to agent-centered search methods with this property in the following as LRTA*-like real-time heuristic search methods. The state values of LRTA* approximate the goal distances of the states. They can be initialized using a heuristic function, such as the straight-line distance between a location and the goal location on a map, which focuses planning towards a goal state. LRTA* and LRTA*-like real-time search methods improve on earlier agent-centered search methods that also used heuristic functions to focus planning but were not guaranteed to terminate (Doran 1967). A longer overview of LRTA*-like real-time search methods is given in (Ishida 1997) and current research issues are outlined in (Koenig 1998).

Figure 3 illustrates the behavior of LRTA* using a simplified goal-directed navigation problem in known terrain without uncertainty about the initial location. The robot can move one location (cell) to the north, east, south, or west, unless that location is untraversable. All action costs are one. The robot has to navigate to the given goal location and then stop. In this case, the states correspond to locations, and the current state corresponds to the current location of the robot. The state values are initialized with the Manhattan distance, that is, the goal distance of the corresponding location if no obstacles were present. For example, the Manhattan distance of the start state C1 is three. Figure 4 visualizes the value surface formed by the initial state values. Notice that a robot does not reach the goal state if it always moves to the successor state with the smallest value and thus performs steepest descent on the initial value surface. It moves back and forth between locations C1 and C2 and thus gets trapped in the local minimum of the value surface at location C2. There are robot-navigation methods that use value surfaces in form of potential fields for goal-directed navigation, often combined with randomized movements to escape the local minima (Arkin 1998). LRTA* avoids this problem by increasing the state values to fill the local minima in the value surface. Figure 5 shows how LRTA* performs a search around the current state of the robot to determine which action to execute next if it breaks ties among actions in the following order: north, east, south, and west. It operates as follows:

1. (Search Step:) LRTA* decides on the local search space. The local search space can be any set of nongoal states that contains the current state (Barto *et al.* 1995). LRTA* typically uses forward search to select a continuous part of the state space around the current state of the agent. For example, it could use A* to determine the local search space. This would make it an on-line variant of A* since LRTA* then interleaves in-



state values before the value-calculation step
state values after the value-calculation step
start location
goal location
= robot location
= local search space

Figure 6: Example with a Larger Local Search Space

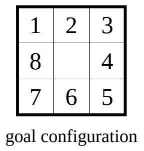


Figure 7: Eight Puzzle

complete A* searches from the current state of the agent with plan executions. Some researchers have also explored versions of LRTA* that do not perform agent-centered search, for example in the context of reinforcement learning with the DYNA architecture (Sutton 1990; Moore and Atkeson 1993). In the example of Figure 3, the local search spaces are minimal, that is, contain only the current state. In this case, LRTA* can construct a search tree around the current state. The local search space consist of all nonleaves of the search tree. Figure 5 shows the search tree for deciding which action to execute in the initial location.

2. (Value-Calculation Step:) LRTA* assigns each state in the local search space its correct goal distance under the assumption that the values of the states just outside of the local search space correspond to their correct goal distances. In other words, it assigns each state in the local search space the minimum of the execution cost for getting from it to a state just outside of the local search space plus the estimated remaining execution cost for getting from there to a goal location, as given by the value of the state just outside of the local search space. Since this lookahead value is a more accurate estimate of the goal distance of the state in the local search space, LRTA* stores it in memory, overwriting the existing value of the state. In the example, the local search space is minimal and LRTA* can simply update the value of the state in the local search space according to the following rule provided that it ignores all actions that can leave the current state unchanged. LRTA* first assigns each leaf of the search tree the value of the corresponding state. The leaf that represents B1 is assigned a value of four and the leaf that represents C2 is assigned a value of two. This step is marked (1) in Figure 5. The new value of the root node C1 then is the minimum of the values of its children plus one, because LRTA* chooses moves that minimize the goal distance and the robot has to execute one additional action to reach the child (2). This value is then stored in memory for C1 (3). Figure 6 shows the result of one value-calculation step for a different example where the local search space is non-minimal.

3. (Action-Selection Step:) LRTA* selects an action for execution that is the beginning of a plan that promises to minimize the execution cost from the current state to a goal state (ties can be broken arbitrarily). In the example, LRTA* selects the action that moves to a child of the root node of the search tree that minimizes the value of the child plus one. Since the estimated execution cost from the current state to a goal state is three when moving east (namely, one plus two) and five when moving north (one plus four), LRTA* decides to move east.

4. (Action-Execution Step:) LRTA* executes the selected action, updates the state of the robot, and repeats the overall process from the new state of the robot until the robot reaches a goal state.

The left column of Figure 3 shows the result of the first couple of steps of LRTA* for the example. The values in parentheses are the new state values calculated by the value-calculation step because the corresponding states are part of the local search space. The robot reaches the goal location after nine action executions.

If there are no goal states, then LRTA* is guaranteed to visit all states repeatedly if the state space is finite and strongly connected, that is, where every state can be reached from every other state. Strongly connected state spaces guarantee that the agent can still reach every state no matter which actions it has executed in the past. This property of LRTA* is important for *covering terrain* (visiting all locations) once or repeatedly such as for *lawn mowing, mine sweeping, and surveillance*. If there are goal states, then LRTA* is guaranteed to reach a goal state in state spaces that are finite and safely explorable, that is, where the agent can still reach a goal state no matter which actions it has executed in the past. This property of LRTA* is important for *goal-directed navigation* (moving to a goal location).

An analysis of the execution cost of LRTA* until it reaches a goal state and how it depends on the informedness of the initial state values and the topology of the state space is given in (Koenig and Simmons 1995; Koenig and Simmons 1996a). This analysis yields insights into when agent-centered search methods solve planning tasks in deterministic domains efficiently. For example, LRTA* tends to be more efficient the more informed the initial state values are and thus the more the initial state values focus the search well, although this correlation is not perfect (Koenig 1998). LRTA* also tends to be more efficient the smaller the average goal distance of all states is. Consider, for example, sliding-tile puzzles, which are sometimes considered to be hard search problems because they have a small goal density. Figure 7, for example, shows the eight puzzle, a sliding-tile puzzle with 181,440 states but only one goal state. However, the average goal distance of the eight puzzle is only 21.5 and its maximal goal distance is only 30 (Reinefeld 1993). This implies that LRTA* can never move far away from the goal state even if it makes a mistake and executes an action that does not decrease the goal distance, which makes the eight puzzle state space easy to search relative to other domains with the same number of states.

If the initial state values are not completely informed and the local search spaces are small, then it is unlikely that the execution cost of LRTA* is minimal. In Figure 3, for example, the robot could reach the goal location in seven action executions. However, LRTA* improves its execution cost, although not necessarily monotonically, as it solves planning tasks with the same goal states in the same state spaces until its execution cost is minimal, under the following conditions: its initial state values are admissible (that is, do not overestimate the goal distances) and it maintains the state values between planning tasks. If LRTA* breaks ties always in the same way, then it eventually keeps following the same minimal-cost path from a given start state. If it breaks ties randomly, then it eventually discovers all minimal-cost paths from the given start state. Thus, LRTA* can always have a small sum of planning and execution cost and still minimize the execution cost in the long run.

Figure 3 (all columns) illustrates this aspect of LRTA*. In the example, LRTA* breaks ties among successor states in the following order: north, east, south, and west. Eventually, the robot always follows a minimal-cost path to the goal location. LRTA* is able to improve its execution cost by making the state values better informed. Figure 8 visualizes the value surface formed by the final state values. The robot now reaches the goal state on a minimal-cost path if it always moves to the successor state with the smallest value (and breaks ties in the order given above) and thus performs steepest descent on the final value surface.

LRTA* always moves in the direction in which it believes the goal state to be. While this might be a good action-selection strategy for reaching the goal state quickly, recent evidence suggests that it might not be a good action-selection strategy for converging to a minimal-cost path quickly. Consequently, researchers have studied LRTA*-like real-time search methods that improve their execution cost faster than LRTA* (Thorpe 1994; Ishida and Shimbo 1996; Edelkamp 1997; Furcy and Koenig 2000). For example, while LRTA* focuses its value updates on what it believes

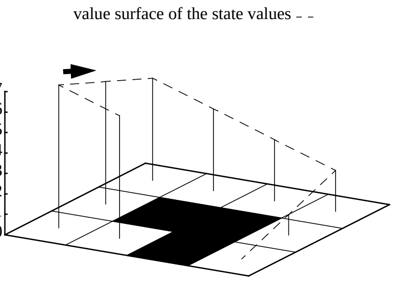


Figure 8: Value Surface after Convergence

to be a minimal-cost path from its *current* state to a goal state, FAst Learning and CONverging Search (FALCONS) (Furcy and Koenig 2000) focuses its value updates on what it believes to be a minimal-cost path from the *start* state to a goal state, and often finds minimal-cost paths faster than LRTA* in undirected state spaces.

In the following sections, we discuss the application of agent-centered search methods to deterministic and nondeterministic planning tasks, and relate these agent-centered search methods to LRTA*.

Deterministic Domains

In deterministic domains, the outcomes of action executions can be predicted with certainty. Many traditional domains from artificial intelligence are deterministic, including sliding-tile puzzles and blocks worlds. Agent-centered search methods can solve off-line planning tasks in these domains by moving a fictitious agent in the state space (Dasgupta *et al.* 1994). In this case, the local search spaces are not imposed by information limitations. Agent-centered search methods thus provide alternatives to traditional search methods, such as A*. They have, for example, successfully been applied to *optimization* and *constraint-satisfaction* problems and are often combined with random restarts. Examples include hill climbing, simulated annealing, tabu search, some SAT-solution methods, and some scheduling methods (Selman 1995; Aarts and Lenstra 1987; Gomes *et al.* 1998). Agent-centered search methods have also been applied to *traditional search* problems (Korf 1990) and *STRIPS-type planning* problems (Bonet *et al.* 1997). For instance, LRTA*-like real-time search methods easily determine plans for the twenty-four puzzle, a sliding-tile puzzle with more than 10^{24} states (Korf 1993), and blocks worlds with more than 10^{27} states (Bonet *et al.* 1997). For these planning problems, agent-centered search methods compete with other heuristic search methods such as greedy (best-first) search (Russell and Norvig 1995), that can find plans faster than agent-centered search, or linear-space best-first search (Russell 1992; Korf 1993), that can consume less memory (Korf 1993; Bonet and Geffner 2001).

Nondeterministic Domains

Many domains from robotics, control, and scheduling are nondeterministic. Planning in nondeterministic domains is often more difficult than planning in deterministic domains since their information limitation can only be overcome by enumerating all possible contingencies, which results in large search spaces. Consequently, it is even more important that agents take their planning cost into account to solve planning tasks efficiently. Agent-centered search in nondeterministic domains has an additional advantage compared to agent-centered search in deterministic domains, namely that it allows agents to gather information early. This is an enormous strength of agent-centered search because this information can be used to resolve some of the uncertainty and thus reduce the amount of planning performed for unencountered situations. Without interleaving planning and plan execution, an agent has to determine a complete conditional plan that solves the planning task, no matter which contingencies arise during its execution. Such a plan can be large. When interleaving planning and plan execution, on the other hand, the agent does not need to plan for every possible contingency. It has to determine only the beginning of a complete plan. After the execution of this subplan, it can observe the resulting state and then repeat the process from the state that actually resulted from the execution of the subplan instead of all states that could have resulted from its execution. We have already described this advantage of agent-centered search in the context of game playing. In the following, we illustrate the same advantage in the context of mobile robotics. Consider, for example, a mobile robot that has to localize itself, that is, to gain certainty about its location. As it moves in the terrain, it can acquire additional information about its current environment via sensing. This information reduces its uncertainty about its location, which makes planning more efficient. Thus, sensing during plan execution and using the acquired knowledge for replanning, often called sensor-based planning (Choset and Burdick 1995), is one way to make the localization problem tractable.

Mobile robots are perhaps the class of agents that have been studied the most, and agent-centered search methods have been used as part of several independently developed robot architectures that robustly perform real-world navigation tasks in structured or unstructured terrain. Navigation often has to combine path planning with map building or localization (Nehmzow 2000). Consequently, we study two different navigation tasks. First, we discuss exploration (map building) and goal-directed navigation in initially unknown terrain but without uncertainty about the initial location. Then, we discuss localization and goal-directed navigation in known terrain with uncertainty about the initial location. Agent-centered search methods have also been used in other nondeterministic domains from mobile robotics, including *moving-target search*, the task of catching moving prey (Ishida and Korf 1991; Ishida 1992; Koenig and Simmons 1995).

We are only interested in the navigation strategy of the robots (not precise trajectory planning). We therefore attempt to isolate the agent-centered search methods from the overall robot architectures, which makes it sometimes necessary to simplify the agent-centered search methods slightly. We assume initially that there is no actuator or sensor noise and that every location can be reached from every other location. All of the following agent-centered search methods are guaranteed to solve the navigation tasks under these assumptions. Since the assumptions are quite strong, we discuss in a later section how to relax them.

Exploration of Unknown Terrain

We first discuss *exploration (map building)* and *goal-directed navigation in initially unknown terrain without uncertainty about the initial location*. The robot does not know a map of the terrain. It can move one location to the north, east, south, or west, unless that location is untraversable. All action costs are one. On-board sensors tell the robot in every location which of the four adjacent locations (north, east, south, west) are untraversable and, for goal-directed navigation, whether the current location is a goal location. Furthermore, the robot can identify the adjacent locations when it observes them again at a later point in time. This assumption is realistic, for example, if dead-reckoning works perfectly, the locations look sufficiently different, or a global positioning system is available. For exploration, the robot has to visit all locations and then stop. For goal-directed navigation, the robot has to navigate to the given goal location and then stop.

The locations (and how they connect) form the initially unknown state space. Thus, the states correspond to locations and the current state corresponds to the current location of the robot. Although all actions have deterministic effects, the planning task is nondeterministic because the robot cannot predict the outcomes of its actions in the unknown part of the terrain. For example, it cannot predict whether the location in front of it will be traversable after it moves forward in unknown terrain. This information limitation is hard to overcome since it is prohibitively time-consuming to enumerate all possible obstacle configurations in the unknown part of the terrain. This problem can be avoided by restricting planning to the known part of the terrain, which makes the planning tasks deterministic and thus efficient to solve. In this case, agent-centered search methods for deterministic state spaces, such as LRTA*, can be used unchanged for exploration and goal-directed navigation in initially unknown terrain.

We now discuss several of these agent-centered search methods. They all impose grids over the terrain. However, they could also use Voronoi diagrams or similar graph representations of the terrain (Latombe 1991). Although they have been developed independently by different researchers, they are all similar to LRTA*, which has been used to transfer analytical results among them (Koenig 1999). They differ in two dimensions, namely how large their local search spaces

Ant robots are simple robots with limited sensing and computational capabilities. They have the advantage that they are easy to program and cheap to build. This makes it feasible to deploy groups of ant robots and take advantage of the resulting fault tolerance and parallelism (Brooks and Flynn 1989). Ant robots cannot use conventional planning methods due to their limited sensing and computational capabilities. To overcome these limitations, ant robots can use LRTA*-like real-time search methods (such as LRTA* or Node Counting) to leave markings in the terrain that can be read by the other ant robots, similar to what real ants do (Adler and Gordon 1992). Ant robots that each run the same LRTA*-like real-time search method on the shared markings (where the locations correspond to states and the markings correspond to state values) cover terrain once or repeatedly even if they move asynchronously, do not communicate with each other except via the markings, do not have any kind of memory, do not know the terrain, cannot maintain maps of the terrain, nor plan complete paths. The ant robots do not even need to be localized, which completely eliminates solving difficult and time-consuming localization problems. The ant robots robustly cover terrain even if they are moved without realizing that they have been moved (say, by people running into them), some ant robots fail, and some markings get destroyed (Koenig *et al.* 2001b). This concept has not yet been implemented on robots although mobile robots have been built that leave markings in the terrain. However, so far these markings have only been short-lived such as odor traces (Russell *et al.* 1994), heat traces (Russell 1997), or alcohol traces (Sharpe and Webb 1998).

Ant Robotics

are and whether their initial state values are uninformed or partially informed:

Sizes of the Local Search Spaces: We call the local search spaces of agent-centered search methods for deterministic state spaces maximal in unknown state spaces if they contain all of the known part of the state space, for example, all visited states. We call the local search spaces minimal if they contain only the current state.

Informedness of the Initial State Values: Heuristic functions that can be used to initialize the state values are often unavailable for exploration and for goal-directed navigation if the coordinates of the goal location are unknown, such as when searching for a post office in an unknown city. Otherwise, the Manhattan distance of a location can be used as an approximation of its goal distance.

In the following, we discuss the three of the four resulting combinations that have been used on robots:

- **Approach 1:** Uninformed LRTA* with minimal local search spaces can be used unchanged for exploration and goal-directed navigation in initially unknown terrain and, indeed, LRTA*-like real-time search methods have been used for this purpose.

  Several LRTA*-like real-time search methods differ from LRTA* with minimal local search spaces only in their value-calculation step (Korf 1990; Russell and Wefald 1991; Thrun 1992; Wagner *et al.* 1997). Consider, for example, Node Counting, an LRTA*-like real-time search method that always moves the robot from its current location to that adjacent location that it has visited the smallest number of times so far. It has been used for exploration by several researchers, either in pure or modified form (Pirzadeh and Snyder 1990; Thrun 1992; Balch and Arkin 1993). For example, it is similar to Avoiding the Past (Balch and Arkin 1993), that has been used on a nomad-class Denning mobile robot that placed well in AAAI autonomous robot competitions. Avoiding the Past differs from Node Counting in that it sums over vectors that point away from locations that are adjacent to the robot with a magnitude that depends on how often these locations have been visited so far, which simplifies its integration into schema-based robot architectures (Arkin 1998). It has also been suggested that Node Counting mimics the exploration behavior of ants (Wagner *et al.* 1999) and can thus be used to build *ant robots* (Koenig *et al.* 2001b), see the side bar.

  Node Counting and uninformed LRTA* with minimal local search spaces differ only in their value-calculation step (if all action costs are one). The state values of Node Counting count how often the states have been visited. Consequently, Node Counting moves the robot to states that have been visited fewer and fewer number of times with the planning objective of getting it as fast as possible to a state that has not been visited at all, that is, an unvisited state (where the robot gains information). The state values of uninformed LRTA*, on the other hand, approximate the distance of the states to a closest unvisited state. Consequently, LRTA* moves the robot to states that are closer and closer to unvisited states with the planning objective of getting it as fast as possible to an unvisited state. Experimental evidence suggests that Node Counting and uninformed LRTA* with minimal local search spaces perform about equally well in many (but not all) domains. However, it is also known that LRTA* can have advantages over Node Counting. For example, it has a much smaller execution cost in the worst case, can use heuristic functions to focus its search, and improves its execution cost as it solves similar planning tasks. An analysis of the execution cost of Node Counting is given in (Koenig and Simmons 1996a; Koenig and Szymanski 1999).

- **Approach 2:** Uninformed LRTA* with maximal local search spaces can be used unchanged for exploration and goal-directed navigation in initially unknown terrain. It results in the following behavior of a robot that has to explore unknown terrain. The robot always moves from its current location with minimal execution cost to an unvisited location (where it gains information), until it has explored all of the terrain (Greedy Mapping). It has been used on a nomad-class tour-guide robot that offered tours to museum visitors (Thrun *et al.* 1998). An analysis of the execution cost of uninformed LRTA* with maximal local search spaces is given in (Koenig 1999; Koenig *et al.* 2001c).

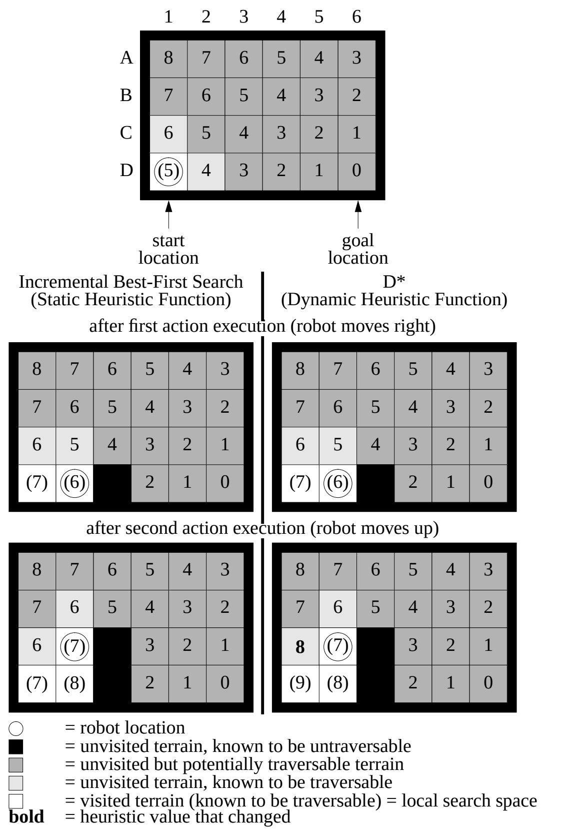

start location
goal location
Incremental Best-First Search (Static Heuristic Function)
D* (Dynamic Heuristic Function)
after first action execution (robot moves right)
after second action execution (robot moves up)
○ = robot location
■ = unvisited terrain, known to be untraversable
▨ = unvisited but potentially traversable terrain
□ = unvisited terrain, known to be traversable
□ = visited terrain (known to be traversable) = local search space
bold = heuristic value that changed

Figure 9: Exploration with Maximal Local Search Spaces

- **Approach 3:** Partially informed LRTA* with maximal local search spaces can be used unchanged for goal-directed navigation in initially unknown terrain. This approach has been called incremental best-first search (Pemberton and Korf 1992). It results in the following behavior of a robot that has to move to a goal location in unknown terrain. It always moves from its current location to an unvisited location (where it gains information) so that it minimizes the sum of the execution cost for getting from its current location to the unvisited location and the estimated remaining execution cost for getting from the unvisited location to the goal location, as given by the value of the unvisited location, until it has reached the goal location.

The heuristic function of incremental best-first search can also be changed dynamically as parts of the terrain get discovered. D* (Stentz 1995) and D* Lite (Likhachev and Koenig 2000), for example, exhibit the following behavior: The robot repeatedly moves from its current location with minimal execution cost to a goal location, assuming that unknown terrain is traversable. (Other assumptions are possible.) When it observes during plan execution that a particular location is untraversable, it corrects its map, uses the updated map to recalculate a minimal-cost path from its current location to the goal location (again making the assumption that unknown terrain is traversable), and repeats this procedure until it reaches the goal location. D* is an example of an assumptive planning method (Nourbakhsh 1997) that exhibits optimism in the face of uncertainty (Moore and Atkeson 1993) because the path that it determines can be traversed only if it is correct in its assumption that unknown terrain is traversable. If the assumption is indeed correct, then the robot reaches the goal location. If the assumption is incorrect, then the robot discovers at least one untraversable location that it did not know about and thus gains information. D* has been used on an autonomous high-mobility multi-wheeled vehicle (HMMWV) that navigated 1,410 meters to the goal location in an unknown area of flat terrain with sparse mounds of slag as well as trees, bushes, rocks, and debris (Stentz and Hebert 1995).

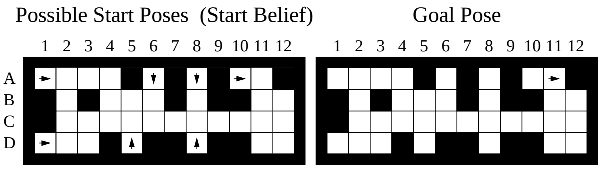


Figure 10: Navigation Task with Unknown Initial Pose

D* is similar to incremental best-first search with the exception that it changes the heuristic function dynamically, which requires it to have initial knowledge about the possible connectivity of the graph, for example, geometrical knowledge of a two-dimensional terrain. Figure 9 illustrates this difference between D* and incremental best-first search. In the example, D* changes the state value of location C1 (even though this location is still unvisited and thus has not been part of any local search space) when it discovers that locations C3 and D3 are untraversable because the layout of the environment implies that it takes now at least eight moves to reach the goal location instead of the six moves suggested by the heuristic function. Dynamically recomputing the heuristic function makes it better informed but takes time, and the search is no longer restricted to the part of the terrain around the current location of the robot. Thus, different from incremental best-first search, D* is not an agent-cetnered search method and its searches are not restricted to the known part of the terrain, which results in an information limitation. D* avoids this problem by making assumptions about the unknown terrain, which makes the planning tasks again deterministic and thus efficient to solve. D* shares with incremental best-first search that it improves its execution cost as it solves planning tasks with the same goal states in the same state spaces until it follows a minimal-cost path to a goal state under the same conditions described for LRTA*.

Robot Localization in Known Terrain

We now discuss *localization* and *goal-directed navigation in known terrain with uncertainty about the initial location*. We illustrate these navigation tasks with a similar scenario as before. Figure 10 shows a simplified example of a goal-directed navigation task in a grid world. The robot knows the map of the terrain, but is uncertain about its start pose, where a pose is a location and orientation (north, east, south, west). It can move forward one location (unless that location is untraversable), turn left ninety degrees, or turn right ninety degrees. All action costs are one. On-board sensors tell the robot in every pose which of the four adjacent locations (front, left, behind, right) are untraversable. For localization, the robot has to gain certainty about its pose and then stop. For goal-directed navigation, the robot has to navigate to a given goal pose and then stop. Since there might be many poses that produce the same sensor reports as the goal pose, this task includes localizing the robot so that it knows that it is in the goal pose when it stops. We assume that localization is possible, which implies that the environment is not completely symmetrical. This modest (and realistic) assumption allows the robot to localize itself and, for goal-directed navigation, then move to the goal pose.

Analytical results about the execution cost of planning methods are often about their worst-case execution cost (here: execution cost for the worst possible start pose) rather than their average-case execution cost. In this case, the states of the localization and goal-directed navigation problems with uncertainty about the initial pose correspond to sets of poses (belief states), namely the poses that the robot could possibly be in. The current state corresponds to the poses that the robot could currently be in. For example, if the robot has no knowledge of its start pose for the goal-directed navigation task shown in Figure 10 but observes walls all around it except in its front, then its start state contains the following seven poses: A 1 $\rightarrow$, A 6 $\downarrow$, A 8 $\downarrow$, A 10 $\rightarrow$, D 1 $\rightarrow$, D 5 $\uparrow$, and D 8 $\uparrow$. Although all actions have deterministic effects, the planning task is nondeterministic because the robot cannot predict the outcomes of its actions with certainty since it is uncertain about its pose. For example, it cannot predict whether the location in front of it will be traversable after it moves forward for the goal-directed navigation task shown in Figure 10. (If its start pose were A 1 $\rightarrow$ then it would see a traversable location in front of it after it moves forward. On the other hand, if its start pose were A 10 $\rightarrow$ then it would see an untraversable location in front of it.) This information limitation can only be overcome by enumerating all possible observations, which results in large search spaces. For example, solving localization tasks with minimal worst-case execution cost is NP-hard, even within a logarithmic factor (Dudek *et al.* 1995; Tovey and Koenig 2000). This analytical result is consistent with empirical results that indicate that performing a complete minimax (and-or) search to determine plans with minimal worst-case execution cost is often completely infeasible (Nourbakhsh 1997). This problem can be avoided in the same way as for exploration and goal-directed navigation in initially unknown terrain, namely by restricting the search spaces, possibly even to the deterministic part of the state space around the current state, which makes the planning tasks efficient to solve. Different from exploration

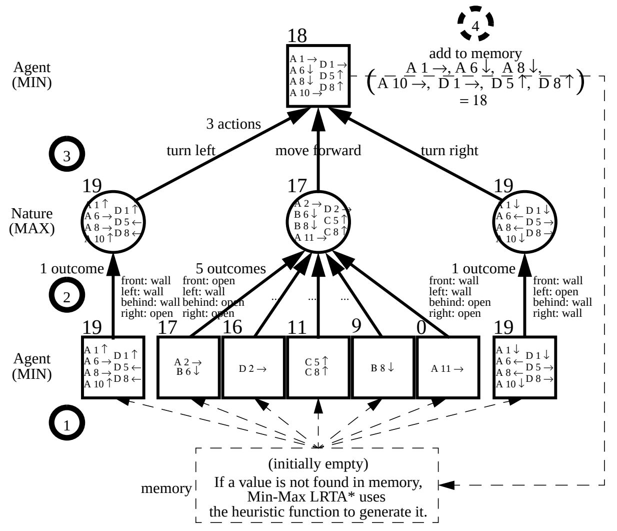


Figure 11: Min-Max LRTA*

and goal-directed navigation in initially unknown terrain, however, agent-centered search methods for deterministic state spaces cannot be used completely unchanged to solve localization and goal-directed navigation tasks with uncertainty about the initial pose. This is so because the robot can no longer predict the outcomes of all actions in its current state with certainty.

To solve this problem, we introduce Min-Max Learning Real-Time A* (Min-Max LRTA*) (Koenig and Simmons 1995; Koenig 2001), a generalization of LRTA* to nondeterministic domains that attempts to minimize the worst-case execution cost. Min-Max LRTA* has been shown to solve simulated navigation tasks efficiently in typical grid worlds (Koenig and Simmons 1998a) and has also been applied to other planning tasks (Bonet and Geffner 2000). It can be used to search not only the deterministic part of the state space around the current state but also larger and thus nondeterministic local search spaces. It treats the navigation tasks as games by assuming that the agent selects the actions and a fictitious opponent, called nature, chooses the resulting observations. Figure 11 (excluding the dashed part) shows how Min-Max LRTA* performs a minimax search around the current belief state of the robot to determine which action to execute next. It operates as follows:

1. (Search Step:) Min-Max LRTA* decides on the local search space. The local search space can be any set of nongoal states that contains the current state. Min-Max LRTA* typically uses forward search to select a continuous part of the state space around the current state of the agent. In the example of Figure 11, the local search space is minimal, that is, contains only the current state. In this case, Min-Max LRTA* can construct a minimax tree around the current state. The local search space consist of all nonleaves of the minimax tree where it is the turn of the agent to move.

2. (Value-Calculation Step:) Min-Max LRTA* calculates for each state in the local search space its correct minimax goal distance under the assumption that the heuristic function determines the correct minimax goal distances for the states just outside of the local search space. The minimax goal distance of a state is the execution cost needed to solve the planning task from this state under the assumption that Min-Max LRTA* attempts to get to a goal state as quickly as possible, nature attempts to prevent it from getting there, and nature does not make mistakes. In the example, the local search space is minimal and Min-Max LRTA* can use a simply minimax search to update the value of the state in the local search space provided that it ignores all actions that can leave the current state unchanged. Min-Max LRTA* first assigns all leaves of the minimax tree the value determined by the heuristic function for the corresponding state. This step is marked (1) in Figure 11. For example, the minimax goal distance of a belief state can be approximated as follows for goal-directed navigation tasks, thereby generalizing the concept of heuristic functions from deterministic to nondeterministic domains: The robot determines for each pose in the belief state how many actions it would have to execute to reach the goal pose if it knew that it was currently in that pose. The calculation of these values involves no uncertainty about the current pose and can be performed efficiently with traditional search methods in the deterministic state space of poses (that is, the known map). The maximum of these values is an approximation of the minimax goal distance of the belief state. This value is 18 for the start belief state used earlier, namely the maximum of 18 for A 1 $\rightarrow$, 12 for A 6 $\downarrow$, 10 for A 8 $\downarrow$, 1 for A 10 $\rightarrow$, 17 for D 1 $\rightarrow$, 12 for D 5 $\uparrow$, and 9 for D 8 $\uparrow$. Min-Max LRTA* then backs up these values towards the root of the minimax tree. The value of a node where it is the turn of nature to move is the maximum of the values of its children since nature chooses moves that maximize the minimax goal distance (2). The value of a node where it is the turn of the agent to move is the minimum of the values of its children plus one, because Min-Max LRTA* chooses moves that minimize the minimax goal distance and the robot has to execute one additional action to reach the child (3).
3. (Action-Selection Step:) Min-Max LRTA* selects an action for execution that is the beginning of a plan that promises to minimize the worst-case execution cost from the current state to a goal state (ties can be broken arbitrarily). In the example, Min-Max LRTA* selects the action that moves to a child of the root node of the minimax search tree that minimizes the value of the child plus one. Consequently, it decides to move forward.
4. (Action-Execution Step:) Min-Max LRTA* executes the selected action (possibly already planning action sequences in response to the possible observations it can make next), makes an observation, updates the belief state of the robot based on this observation, and repeats the overall process from the new belief state of the robot until the navigation task is solved.

Min-Max LRTA* has to ensure that it does not cycle forever. It can randomize its action-selection process or use one of the following two approaches to gain information between plan executions and thus guarantee progress:

- **Direct Information Gain:** If Min-Max LRTA* uses sufficiently large local search spaces, then it can determine plans that guarantee, even in the worst case, that their execution results in a reduction of the number of poses that the robot could possibly be in and thus in an information gain (Greedy Localization). For example, moving forward reduces the number of possible poses from seven to at most two for the goal-directed navigation task shown in Figure 10. Min-Max LRTA* with direct information gain is similar to the behavior of the Delayed Planning Architecture with the viable plan heuristic (Nourbakhsh 1997). The Delayed Planning Architecture has been used by their authors on Nomad 150 mobile robots in robot programming classes to navigate mazes that were built with three-foot high, forty inch long cardboard walls. The size of the mazes was limited only by the space available.
- **Indirect Information Gain:** Min-Max LRTA* with direct information gain does not apply to all planning tasks. Even if it applies, as is the case for the navigation tasks with uncertainty about the initial pose, the local search spaces and thus the planning cost that it needs to guarantee a direct information gain can be large. To operate with smaller local search spaces, it can use LRTA*-like real-time search. It then operates as before, with the following two changes: First, when Min-Max LRTA* needs the values of a state just outside of the local search space (that is, the value of a leaf of the minimax tree) in the value-calculation step, it now checks first whether it has already stored a value for this state in memory. If so, then it uses this value. If not, then it calculates the value using the heuristic function, as before. Second, after Min-Max LRTA* has calculated the value of a state in the local search space where it is the turn of the agent to move, it now stores it in memory, overwriting any existing value of the corresponding state (4). Figure 11 (including the dashed part) summarizes the steps of Min-Max LRTA* with indirect information gain before it decides to move forward.

  An analysis of the execution cost of Min-Max LRTA* with indirect information gain is given in (Koenig and Simmons 1995). It is an extension of the corresponding analysis of LRTA* since Min-Max LRTA* with indirect information gain reduces in deterministic domains to LRTA*. This is so because Min-Max LRTA* basically uses the largest value of all potential successor states that can result from the execution of a given action in a given state at those places in the value-calculation and action-selection steps where LRTA* simply uses the value of the only successor state.

  The increase of the state values can be interpreted as an indirect information gain that guarantees that Min-Max LRTA* reaches a goal state in finite state spaces where the minimax goal distance of every state is finite (a generalization of safely explorable state spaces to nondeterministic domains). A disadvantage of Min-Max LRTA* with indirect information gain over Min-Max LRTA* with direct information gain is that the robot has to store potentially one value in memory for each state it has visited. In practice, however, the memory requirements of LRTA*-like real-time search methods often seem to be small, especially if the initial state values are well informed and thus focus the search, which prevents them from visiting a large number of states. Furthermore, LRTA*-like real-time search methods only need to store the values of those states in memory that differ from the initial state values. If the values are the same, then they can be automatically re-generated when they are not found in memory. For the example from Figure 11, for instance, it is unnecessary to store the calculated value 18 of the initial belief state in memory. An advantage of Min-Max LRTA* with indirect information gain over Min-Max LRTA* with direct information gain is that it is able to operate with smaller local search spaces, even local search spaces that contain only the current state. Another advantage is that it improves its execution cost, although not necessarily monotonically, as it solves localization and goal-directed navigation tasks with uncertainty about the initial pose in the same terrain but possibly different start poses, under the following conditions: its initial state values do not overestimate the minimax goal distances and it maintains the state values between planning tasks. The state values converge after a bounded number of mistakes, where it counts as one mistake when Min-Max LRTA* reaches a goal state with an execution cost that is larger than the minimax goal distance of the start state. After convergence, its execution cost is at most as large as the minimax goal distance of the start state. Although Min-Max LRTA* typically needs to solve planning tasks multiple times to minimize the execution cost, it might still be able to do so faster than one complete minimax (and-or) search if nature is not as malicious as a minimax search assumes and some successor states do not occur in practice, for example, when (unknown to the robot) not all poses occur as start poses for localization tasks. Min-Max LRTA* does not plan for these situations since it only plans for situations that it actually encounters.

Notice the similarity between Min-Max LRTA* and the minimax search method used by game-playing programs. Even the reasons why agent-centered search is well suited are similar for both planning tasks. In both cases, the state spaces are too large to perform complete searches in a reasonable amount of time. There are a large number of goal states and thus no unique starting point for a backward search. Finally, the state spaces are nondeterministic and the agent thus cannot control the course of events completely. Consequently, plans are really trees with a unique starting point (root) for a forward search but no unique starting point (leaves) for backward search. Despite these similarities, however, Min-Max LRTA* differs from a minimax search in two aspects. First, Min-Max LRTA* assumes that all terminal states (states where the planning task is over) are desirable and attempts to get to a terminal state fast. Minimax search, on the other hand, distinguishes terminal states of different quality (wins and losses) and attempts to get to a winning terminal state. It is not important how many moves it takes to get there, which changes the semantics of the values calculated by the heuristic functions and how the values get backed up towards the root of the minimax tree. Second, Min-Max LRTA* with indirect information gain changes its evaluation function during planning and thus needs to store the changed values in memory.

Generalizations of Agent-Centered Search

In the following, we briefly discuss how to relax some of the assumptions made so far.

- **Irreversible actions:** We have assumed that the agent can recover from the execution of each action. If this is not the case, then the agent has to guarantee that the execution of each action does not make it impossible to reach a goal state, which is often possible by increasing the local search spaces of agent-centered search methods. For example, if Min-Max LRTA* is applied to goal-directed navigation tasks with uncertainty about the initial pose and irreversible actions and always determines a plan after whose execution the belief state is guaranteed to contain either only the goal pose, only poses that are part of the current belief state of the robot, or only poses that are part of the start belief state, then either the goal-directed navigation task remains solvable in the worst case or it was not solvable in the worst case to begin with (Nourbakhsh 1997).
- **Uncertainty:** We have assumed that there is no actuator or sensor noise. This is a reasonable assumption in some environments. For example, we mentioned earlier that the Delayed Planning Architecture has been used on Nomad 150 mobile robots for goal-directed navigation with uncertainty about the initial pose in mazes. The success rate of turning left or right was reported as 100.00 percent in these environments, the success rate of moving forward (where possible) was at least 99.57 percent, and the success rate of making the correct observations in all four directions simultaneously was at least 99.38 percent (Nourbakhsh 1996). These large success rates enable one to use agent-centered search methods that assume that there is no actuator or sensor noise, especially since the rare failures are usually quickly noticed when the number of possible poses drops to zero, in which case the robot simply reinitializes its belief state to all possible poses and then continues to use the agent-centered search methods unchanged. In less constrained terrain, however, it is important to take actuator and sensor noise into account, and agent-centered search methods can do so.

  Planning tasks with actuator but no sensor noise can be modeled with totally observable Markov decision process (MDP) problems (Boutilier *et al.* 1999), and can be solved with agent-centered search methods. Consider, for example, Min-Max LRTA* with indirect information gain. It assumes that nature chooses the action outcome that is worst for the agent. The value of a node where it is the turn of nature to move is thus calculated as the maximum of the values of its children, and Min-Max LRTA* attempts to minimize the worst-case execution cost. The assumption that nature chooses the action outcome that is worst for the agent, however, is often too pessimistic and can then make planning tasks wrongly appear to be unsolvable. In such situations, Min-Max LRTA* can be changed to assume that nature chooses action outcomes according to a probability distribution that depends only on the current state and the executed action, resulting in an MDP. In this case, the value of a node where it is the turn of nature to move is calculated as the average of the values of its children weighted with the probability of their occurrence as specified by the probability distribution. Probabilistic LRTA*, this probabilistic variant of Min-Max LRTA*, then attempts to minimize the average execution cost rather than the worst-case execution cost. Probabilistic LRTA* reduces in deterministic domains to LRTA*, just like Min-Max LRTA*. It is a special case of Trial-Based Real-Time Dynamic Programming (RTDP) (Barto *et al.* 1995) that uses agent-centered search and can, for example, be used instead of LRTA* for exploration and goal-directed navigation in unknown terrain with actuator but no sensor noise. There also exist LRTA*-like real-time search methods that attempt to satisfy performance criteria different from minimizing the worst-case or average execution cost (Littman and Szepesvári 1996). MDPs often use discounting, that is, discount an (execution) cost in the far future more than a cost in the immediate future. Discounting thus suggest to concentrate planning on the immediate future, which benefits agent-centered search (Kearns *et al.* 1999).

Two kinds of planning methods are related to Probabilistic LRTA*:

- **Plan-envelope methods:** Plan-envelope methods operate on MDPs and thus have the same planning objective as Probabilistic LRTA* (Bresina and Drummond 1990; Dean *et al.* 1995). Like agent-centered search methods, they reduce the planning cost by searching only small local search spaces (plan envelopes). If the local search space is left during plan execution, then they repeat the overall process from the new state, until they reach a goal state. However, they plan all the way from the start state to a goal state, using local search spaces that usually border at least one goal state and are likely not to be left during plan execution.
- **Reinforcement-learning methods:** Reinforcement learning is learning from rewards and penalties that can be delayed. Reinforcement-learning methods often operate on MDPs and thus have the same planning objective as Probabilistic LRTA* but assume that the probabilities are unknown and have to be learned. Many reinforcement-learning methods use agent-centered search and are similar to LRTA*-like real-time search methods (Koenig and Simmons 1993; Barto *et al.* 1995), which makes it possible to transfer analytical results from LRTA*-like real-time search to reinforcement learning (Koenig and Simmons 1996b). The reason for using agent-centered search in the context of reinforcement learning is the same as the one in the context of exploration, namely that interleaving planning and plan execution allows one to gain new knowledge (that is, learn). An additional advantage in the context of reinforcement learning is that the agent samples probabilities more often in the parts of the state space that it is more likely to encounter (Parr and Russell 1995). Reinforcement learning has been applied to *game playing* (Tesauro 1994), *elevator control* (Crites and Barto 1996), *robot control* including pole balancing and juggling (Schaal and Atkeson 1994), *robot navigation* including wall following (Lin 1993), and similar control tasks. Good overviews of reinforcement learning methods are given in (Kaelbling *et al.* 1996) and (Sutton and Barto 1998).

Planning tasks with actuator and sensor noise can be modeled with partially observable MDPs (POMDPs) (Kaelbling *et al.* 1998). Very reliable robot architectures have used POMDPs for robot navigation in unconstrained terrain. This includes corridor navigation on a nomad-class RWI delivery robot that received navigation requests from users worldwide via the World Wide Web and has traveled over 240 kilometers in a four-year period (Simmons and Koenig 1995; Koenig *et al.* 1996; Koenig and Simmons 1998b; Koenig 1997b). Similar POMDP-based navigation architectures (sometimes also called Markov navigation or Markov localization) have also been explored at Carnegie Mellon University (Burgard *et al.* 1996), Brown University (Cassandra *et al.* 1996), Michigan State University (Mahadevan *et al.* 1998), SRI International (Konolige and Chou 1999), and others, with interesting recent developments (Thrun *et al.* 2001). An overview can be found in (Thrun 2000). POMDPs over world states (for example, poses) can be expressed as MDPs over belief states, where the belief states are now probability distributions over the world states rather than sets of world states. Consequently, POMDPs can be solved with RTDP-BEL (Bonet and Geffner 2000), an application of Probabilistic LRTA* to the discretized belief space, provided that they are sufficiently small. Other solution methods for POMDPs include combinations of Partially Observable Value Approximation (SPOVA) (Parr and Russell 1995) or forward search (Hansen 1998) with agent-centered search, although the application of these agent-centered search methods to robot navigation is an area of current research because the resulting POMDPs are often too large to be solved by current methods. However, they could, in principle, be applied to localization and goal-directed navigation with uncertainty about the initial pose. They differ from the methods that we discussed in that context in that they are more general and have a different (and often preferable) planning objective. On the other hand, their state spaces are typically larger or even continuous.

Properties of Agent-Centered Search

Agent-centered search methods are related to various planning methods. For example, they are related to off-line forward-chaining (progression) planners. Forward-chaining planners can be competitive with backward-chaining (regression), means-ends, and partial-order planners (Bacchus and Kabanza 1995). They have the advantage that the search concentrates on parts of the state space that are guaranteed to be reachable from the current state of the agent. Domain-specific control knowledge can easily be specified for them in a declarative way that is modular and independent of the details of the planning method. They can easily utilize this knowledge because they have complete knowledge of the state at all times. They can also utilize powerful representation languages since it is easier to determine the successor state of a completely known state than the predecessor state of a state (such as the goal) that is only partially known. Most agent-centered search methods share these properties with forward-chaining planners since they use forward search to generate and search the local search spaces.

Agent-centered search methods are also related to on-line planners. The proceedings of the AAAI-97 Workshop on On-Line Search give a good overview of planning methods that interleave planning and plan execution (Koenig *et al.* 1997). For example, there is a large body of theoretical work on robot localization and exploration in the area of theoretical robotics and theoretical computer science. These theoretical planning methods can outperform greedy heuristic planning methods. For example, all agent-centered search methods that we discussed in the context of exploration and goal-directed navigation in unknown terrain, even the ones with maximal local search spaces, have been proven not to minimize the execution cost in the worst case, although their execution costs are small for typical navigation tasks encountered in practice (Koenig and Smirnov 1996; Koenig 1999; Koenig *et al.* 2001a). A similar statement also holds for the agent-centered search methods that we discussed in the context of localization and goal-directed navigation with uncertainty about the initial location (Tovey and Koenig 2000). We explain the success of agent-centered search methods despite this disadvantage with their desirable properties, some of which we list in the following:

- **Theoretical Foundation:** Unlike many existing ad-hoc planning methods that interleave planning and plan execution, many agent-centered search methods have a solid theoretical foundation, that allows one to characterize their behavior analytically. For example, they are guaranteed to reach a goal state under realistic assumptions and their execution cost can be analyzed formally.
- **Anytime Property:** Any-time contract algorithms (Russell and Zilberstein 1991) are planning methods that can solve planning tasks for any given bound on their planning time, and their solution quality increases with the available planning time. Many agent-centered search methods allow for fine-grained control over how much planning to perform between plan executions by varying the sizes of their local search spaces. Thus, they can be used as any-time contract algorithms for determining which action to execute next, which allows them to adjust the amount of planning performed between plan executions to the planning and execution speeds of robots or the time a player is willing to wait for a game-playing program to make a move.
- **Heuristic search control:** Different from chronological backtracking, that can also be used for goal-directed navigation, many agent-centered search methods can use heuristic functions in form of approximations of the goal distances of the states to focus planning which can reduce the planning cost without increasing the execution cost, or reduce the execution cost without increasing the planning cost.
- **Robustness:** Agent-centered search methods are general-purpose (domain-independent) planning methods that seem to work robustly across domains. For example, they can handle uncertainty, including actuator and sensor noise.
- **Simple integration into agent architectures:** Many agent-centered search methods are simple to implement and integrate well into complete agent architectures. Agent-centered search methods are robust towards the inevitable inaccuracies and malfunctions of other architecture components, are reactive to the current situation, and do not need to have control of the agent at all times, which is important because planning methods should only provide advice on how to act and work robustly even if that advice is ignored from time to time (Agre and Chapman 1987). For example, if a robot has to re-charge its batteries during exploration, then it might have to preempt exploration and move to a known power outlet. Once restarted, the robot should be able to resume exploration from the power outlet, instead of having to return to the location where exploration was stopped (which could be far away) and resume its operation from there. Many agent-centered search methods exhibit this behavior automatically.
- **Performance improvement with experience:** Many agent-centered search methods amortize learning over several planning episodes, which allows them to determine a plan with a suboptimal execution cost fast and then improve the execution cost as they solve similar planning tasks, until the execution cost is minimal or satisficing. This is an important property because no planning method that executes actions before their consequences are completely known can guarantee a small execution cost right away, and planning methods that do not improve their execution cost do not behave efficiently in case similar planning tasks unexpectedly repeat. For example, when a mobile robot plans a trajectory for a delivery task it is important that the robot solves the delivery task suf-

ficiently fast, that is, with a small sum of planning and execution cost, which might prevent it from minimizing the execution cost right away. However, if the robot has to solve the delivery task repeatedly, it should be able to follow a minimal-cost path eventually.

- **Distributed Search:** If several agents are available, then they can often solve planning tasks cooperatively by performing an individual agent-centered search each but sharing the search information, thereby reducing the execution cost. For example, off-line planning tasks can be solved on several processors in parallel by running an LRTA*-like real-time search method on each processor and letting all LRTA*-like real-time search methods share their values (Knight 1993). Exploration tasks can be solved with several robots by running an agent-centered search method, such as uninformed LRTA* with maximal local search spaces, on each robot and let them share the maps. More complex exploration schemes are also possible (Simmons *et al.* 1997). Finally, we have already discussed that terrain coverage tasks can be solved with several ant robots by running LRTA*-like real-time search methods on each robot and let them share the markings.

While these properties can make agent-centered search methods the planning methods of choice, it is important to realize that they are not appropriate for every planning task. For example, agent-centered search methods execute actions before their consequences are completely known and thus cannot guarantee a small execution cost when they solve a planning task for the first time. If a small execution cost is important, one might have to perform complete searches before starting to execute actions. Furthermore, agent-centered search methods trade off the planning and execution costs but do not reason about the trade-off explicitly. In particular, it can sometimes be beneficial to update state values that are far away from the current state, and forward searches might not be able to detect these states efficiently. In these cases, one can make use of ideas from limited rationality and reinforcement learning (DYNA), as we have discussed. Finally, some agent-centered search methods potentially have to store a value in memory for each visited state and thus can have large memory requirements if the initial state values do not focus the search well. In some nondeterministic domains, one can address this problem by increasing their lookahead sufficiently, as we have discussed. In other cases, one might have to use search methods that guarantee a small memory consumption, such as linear-space best-first search.

On the other hand, there are also a large number of planning tasks for which agent-centered search methods are well suited, including the navigation tasks discussed in this article. When designing agent-centered search methods, one has to make several design decisions such as how much to plan between plan executions, how many actions to execute between planning, and how to avoid cycling forever.

- **How much to plan between plan executions:** The amount of planning between plan executions can be limited by time constraints or what is known about the domain. Sometimes a larger amount of planning can guarantee that the agent does not execute actions from which it cannot recover and that it makes progress towards a goal state.

  The amount of planning between plan executions also influences the planning and execution costs and thus also the sum of planning and execution cost. Agent-centered search methods with a sufficiently large amount of planning between plan executions perform a complete search without interleaving planning and plan execution and move from the start state with minimal execution cost to a goal state. Typically, reducing the amount of planning between plan executions reduces the (overall) planning cost but increases the execution cost (because the agent-centered search methods select actions based on less information), although theoretically the planning cost could also increase if the execution cost increases sufficiently (because the agent-centered search methods need to plan more frequently). The amount of planning between plan executions that minimizes the sum of planning and execution cost depends on the planning and execution speeds of the agent (Koenig 2000).

  – **Fast-acting agents:** Less planning between plan executions tends to benefit agents whose execution speed is sufficiently fast compared to their planning speed since the resulting increase in execution cost is small compared to the resulting decrease in planning cost, especially if heuristic knowledge focuses planning sufficiently well. For example, the sum of planning and execution cost approaches the planning cost as the execution speed increases, and the planning cost can often be reduced by reducing the amount of planning between plan executions. Agents that are only simulated, such as the fictitious agents discussed in the section on "Deterministic Domains," are examples of fast-acting agents. Since fictitious agents move in almost no time, local search spaces that correspond to lookaheads of only one or two action executions often minimize the sum of planning and execution cost (Korf 1990; Knight 1993).

  – **Slowly-acting agents:** More planning between plan executions is needed for agents whose planning speed is sufficiently fast compared to their execution speed. For example, the sum of planning and execution cost approaches the execution cost as the planning speed increases, and the execution cost can often be reduced by increasing the amount of planning between plan executions. Most robots are examples of slowly-acting agents. Thus, while we used LRTA* with minimal local search spaces in Figure 3 to illustrate how LRTA* works, using small lookaheads is actually not a good idea on robots.

- **How many actions to execute between planning:** Agent-centered search methods can execute actions until they reach a state just outside of the local search space.

They can also stop executing actions at any time after they have executed the first action. Executing more actions typically results in smaller planning costs (because the agent-centered search methods need to plan less frequently), while executing fewer actions typically results in smaller execution costs (because the agent-centered search methods select actions based on more information).

- **How to avoid cycling forever:** Agent-centered search methods have to ensure that they do not cycle without making progress towards a goal state. This is a potential problem since they execute actions before their consequences are completely known. The agent-centered search methods then have to ensure both that it remains possible to achieve the goal and that they eventually do so. The goal remains achievable if no actions exist whose execution makes it impossible to achieve the goal, if the agent-centered search methods can avoid the execution of such actions in case they do exist, or if the agent-centered search methods have the ability to reset the agent into the start state. Actually achieving the goal is more difficult. Often, a sufficiently large amount of planning between plan executions can guarantee an information gain and thus progress. Agent-centered search methods can also store information in memory to prevent cycling forever. LRTA*-like real-time search methods, for example, store a value in memory for each visited state. Finally, it is sometimes possible to break cycles by randomizing the action-selection process slightly, possibly together with resetting the agents into a start state (random restart) after the execution cost has become large.

Conclusions

In this article, we have argued that agent-centered search methods are efficient and broadly applicable planning methods in both single-agent and multi-agent domains including traditional search, STRIPS-type planning, moving-target search, planning with totally and partially observable Markov decision process problems, reinforcement learning, constraint satisfaction, and robot navigation. We illustrated this planning paradigm with several agent-centered search methods that have been developed independently in the literature and have been used to solve real-world planning tasks as part of complete agent architectures.

Acknowledgments

Thanks to David Furcy, Matthias Heger, Richard Korf, Michael Littman, Yaxin Liu, Illah Nourbakhsh, Patrawadee Prasangsit, Reid Simmons, Yury Smirnov, and all student participants in the class on "Modern Approaches to Planning" for helpful discussions, especially Bill Murdock. Special thanks to Richard Korf for many helpful suggestions that improved the initial manuscript dramatically. We ignored his advice to entitle this paper "Real-Time Search" (since not all agent-centered search methods select actions in constant time) but feel bad about it. The Intelligent Decision-Making Group at Georgia Tech is partly supported by an NSF award under contract IIS-9984827. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies or the U.S. government.

References

Aarts, E. and Lenstra, J. 1987. *Local Search in Combinatorial Optimization*. Wiley.

Adler, F. and Gordon, D. 1992. Information collection and spread by networks of patrolling ants. *The American Naturalist* 140(3):373–400.

Agre, P. and Chapman, D. 1987. Pengi: An implementation of a theory of activity. In *Proceedings of the National Conference on Artificial Intelligence*. 268–272.

Arkin, R. 1998. *Behavior-Based Robotics*. MIT Press.

Bacchus, F. and Kabanza, F. 1995. Using temporal logic to control search in a forward chaining planners. In *New Directions in Planning*. 141–153.

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.

Bertsekas, D. and Tsitsiklis, J. 1997. *Parallel and Distributed Computation: Numerical Methods*. Athena Scientific.

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

Bonet, B. and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling*. 52–61.

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

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

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Bresina, J. and Drummond, M. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proceedings of the National Conference on Artificial Intelligence*. 138–144.

Brooks, R. and Flynn, A. 1989. Fast, cheap, and out of control: A robot invasion of the solar system. *Journal of the British Interplanetary Society* 478–485.

Burgard, W.; Fox, D.; Hennig, D.; and Schmidt, T. 1996. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the National Conference on Artificial Intelligence*. 896–901.

Cassandra, A.; Kaelbling, L.; and Kurien, J. 1996. Acting under uncertainty: Discrete Bayesian models for mobile robot navigation. In *Proceedings of the International Conference on Intelligent Robots and Systems*. 963–972.

Choset, H. and Burdick, J. 1995. Sensor-based planning, part II: Incremental construction of the generalized Voronoi graph. In *Proceedings of the International Conference on Robotics and Automation*. 1643–1649.

Crites, R. and Barto, A. 1996. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press.

Dasgupta, P.; Chakrabarti, P.; and DeSarkar, S. 1994. Agent searching in a tree and the optimality of iterative deepening. *Artificial Intelligence* 71:195–208.

Dean, T.; Kaelbling, L.; Kirman, J.; and Nicholson, A. 1995. Planning under time constraints in stochastic domains. *Artificial Intelligence* 76(1–2):35–74.

Doran, J. 1967. An approach to automatic problem-solving. In *Machine Intelligence*, volume 1. Oliver and Boyd. 105–124.

Dudek, G.; Romanik, K.; and Whitesides, S. 1995. Localizing a robot with minimum travel. In *Proceedings of the 6th Annual ACM-SIAM Sympsosium on Discrete Algorithms*. 437–446.

Edelkamp, S. 1997. New strategies in real-time heuristic search. In Koenig, S.; Blum, A.; Ishida, T.; and Korf, R., editors 1997, *Proceedings of the AAAI Workshop on On-Line Search*. 30–35. Available as AAAI Technical Report WS-97-10.

Furcy, D. and Koenig, S. 2000. Speeding up the convergence of real-time search. In *Proceedings of the National Conference on Artificial Intelligence*. 891–897.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence*. 431–437.

Hansen, E. 1998. Solving POMDPs by searching in policy space. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*. 211–219.

Horvitz, E.; Cooper, G.; and Heckerman, D. 1989. Reflection and action under scarce resources: Theoretical principles and empirical study. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1121–1127.

Ishida, T. and Korf, R. 1991. Moving target search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 204–210.

Ishida, T. and Shimbo, M. 1996. Improving the learning efficiencies of real-time search. In *Proceedings of the National Conference on Artificial Intelligence*. 305–310.

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

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

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

Kaelbling, L.; Littman, M.; and Cassandra, A. 1998. Planning and acting in partially observable stochastic domains. *Artificial Intelligence* 101(1–2):99–134.

Kearns, M.; Mansour, Y.; and Ng, A. 1999. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1324–1331.

Knight, K. 1993. Are many reactive agents better than a few deliberative ones? In *Proceedings of the International Joint Conference on Artificial Intelligence*. 432–437.

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 nondeterministic 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. Appeared also as a book chapter in: Recent Advances in Reinforcement Learning; L. Kaelbling (ed.); Kluwer Academic Publishers; 1996.

Koenig, S. and Simmons, R.G. 1998a. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*. 145–153.

Koenig, S. and Simmons, R.G. 1998b. Xavier: A robot navigation architecture based on partially observable Markov decision process models. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., editors 1998b, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*. MIT Press. 91–122.

Koenig, S. and Smirnov, Y. 1996. Graph learning with a nearest neighbor approach. In *Proceedings of the Conference on Computational Learning Theory*. 19–28.

Koenig, S. and Szymanski, B. 1999. Value-update rules for real-time search. In *Proceedings of the National Conference on Artificial Intelligence*. 718–724.

Koenig, S.; Goodwin, R.; and Simmons, R.G. 1996. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In Dorst, L.; Lambalgen, M.van; and Voorbraak, R., editors 1996, *Reasoning with Uncertainty in Robotics*, volume 1093 of *Lecture Notes in Artificial Intelligence*. Springer. 322–337.

Koenig, S.; Blum, A.; Ishida, T.; and Korf, R., editors 1997. *Proceedings of the AAAI-97 Workshop on On-Line Search*. AAAI Press. Available as AAAI Technical Report WS-97-10.

Koenig, S.; Smirnov, Y.; and Tovey, C. 2001a. Performance bounds for planning in unknown terrain. Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia).

Koenig, S.; Szymanski, B.; and Liu, Y. 2001b. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence – Special Issue on Ant Robotics* 31:41–76.

Koenig, S.; Tovey, C.; and Halliburton, W. 2001c. Greedy mapping of terrain. In *Proceedings of the International Conference on Robotics and Automation*. 3594–3599.

Koenig, S. 1996. Agent-centered search: Situated search with small look-ahead. In *Proceedings of the National Conference on Artificial Intelligence*. 1365.

Koenig, S. 1997a. *Chapter 2 of Goal-Directed Acting with Incomplete Information: Acting with Agent-Centered Search*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Koenig, S. 1997b. *Chapter 3 of Goal-Directed Acting with Incomplete Information: Acting with POMDPs*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania).

Koenig, S. 1998. Real-time heuristic search: Research issues. In *Proceedings of the Workshop on Planning as Combinatorial Search: Propositional, Graph-Based, and Disjunctive Planning Methods at the Intenational Conference on Artificial Intelligence Planning Systems*. 75–79.

Koenig, S. 1999. Exploring unknown environments with real-time search or reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 11. 1003–1009.

Koenig, S. 2000. A quantitative comparision of forward and backward search for robot navigation in unknown terrain. Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia). (In preparation).

Koenig, S. 2001. Minimax real-time heuristic search. *Artificial Intelligence – Special Issue on Heuristic Search* 129:165–197.

Konolige, K. and Chou, K. 1999. Markov localization using correlation. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1154–1159.

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.

Latombe, J.-C. 1991. *Robot Motion Planning*. Kluwer Academic Publishers.

Likhachev, M. and Koenig, S. 2000. Improved fast replanning for mobile robots. Technical report, College of Computing, Georgia Institute of Technology, Atlanta (Georgia). (In preparation).

Lin, L.-J. 1993. *Reinforcement Learning for Robots using Neural Networks*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania). Available as Technical Report CMU-CS-93-103.

Littman, M. and Szepesvári, C. 1996. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the International Conference on Machine Learning*. 310–318.

Mahadevan, S.; Theocharous, G.; and Khaleeli, N. 1998. Rapid concept learning for mobile robots. *Machine Learning* 31(1–3):7–27.

Moore, A. and Atkeson, C. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13(1):103–130.

Nehmzow, U. 2000. *Mobile Robotics: A Practical Introduction*. Springer.

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

Nourbakhsh, I. 1996. *Robot Information Packet*. Distributed at the AAAI-96 Spring Symposium on Planning with Infomplete Information for Robot Problems.

Nourbakhsh, I. 1997. *Interleaving Planning and Execution for Autonomous Robots*. Kluwer Academic Publishers.

Parr, R. and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1088–1094.

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

Pemberton, J. and Korf, R. 1992. Incremental path planning on graphs with cycles. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*. 179–188.

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 Norvig, P. 1995. *Artificial Intelligence – A Modern Approach*. Prentice Hall.

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

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

Russell, R.; Thiel, D.; and Mackay-Sim, A. 1994. Sensing odour trails for mobile robot navigation. In *Proceedings of the International Conference on Robotics and Automation*. 2672–2677.

Russell, S. 1992. Efficient memory-bounded search methods. In *Proceedings of the European Conference on Artificial Intelligence*. 1–5.

Russell, R. 1997. Heat trails as short-lived navigational markers for mobile robots. In *Proceedings of the International Conference on Robotics and Automation*. 3534–3539.

Schaal, S. and Atkeson, C. 1994. Robot juggling: An implementation of memory-based learning. *Control Systems Magazine* 14(1):57–71.

Selman, B. 1995. Stochastic search and phase transitions: Ai meets physics. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 998–1002.

Sharpe, R. and Webb, B. 1998. Simulated and situated models of chemical trail following in ants. In *Proceedings of the International Conference on Simulation of Adaptive Behavior*. 195–204.

Simmons, R. and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 1080–1087.

Simmons, R.; Apfelbaum, D.; Burgard, W.; Fox, D.; Moors, M.; Thrun, S.; and Younes, H. 1997. Coordination for multi-robot exploration and mapping. In *Proceedings of the National Conference on Artificial Intelligence*. 852–858.

Stentz, A. and Hebert, M. 1995. A complete navigation system for goal acquisition in unknown environments. *Autonomous Robots* 2(2):127–145.

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

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

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

Tesauro, G. 1994. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6(2):215–219.

Thorpe, P. 1994. A hybrid learning real-time search algorithm. Master's thesis, Computer Science Department, University of California at Los Angeles, Los Angeles (California).

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 Kortenkamp, D.; Bonasso, R.; and Murphy, R., editors 1998, *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*. MIT Press. 21–52.

Thrun, S.; Fox, D.; Burgard, W.; and Dellaert, F. 2001. Robust monte carlo localization for mobile robots. *Artificial Intelligence Journal* 128(1–2):99–141.

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

Thrun, S. 2000. Probabilistic algorithms in robotics. *Artificial Intelligence Magazine* 21(4):93–109.

Tovey, C. and Koenig, S. 2000. Gridworlds as testbeds for planning with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*. 819–824.

Wagner, I.; Lindenbaum, M.; and Bruckstein, A. 1997. On-line graph searching by a smell-oriented vertex process. In Koenig, S.; Blum, A.; Ishida, T.; and Korf, R., editors 1997, *Proceedings of the AAAI Workshop on On-Line Search*. 122–125. Available as AAAI Technical Report WS-97-10.

Wagner, I.; Lindenbaum, M.; and Bruckstein, A. 1999. Distributed covering by ant-robots using evaporating traces. *IEEE Transactions on Robotics and Automation* 15(5):918–933.

Zilberstein, S. 1993. *Operational Rationality through Compilation of Anytime Algorithms*. Ph.D. Dissertation, Computer Science Department, University of California at Berkeley, Berkeley (California).