Minimax Real-Time Heuristic Search
GIT-COGSCI-2001/02

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

Abstract

Real-time heuristic search methods interleave planning and plan executions and plan only in the part of the domain around the current state of the agents. This is the part of the domain that is immediately relevant for the agents in their current situation. So far, real-time heuristic search methods have mostly been applied to deterministic planning tasks. In this article, we argue that real-time heuristic search methods can efficiently solve nondeterministic planning tasks. Planning in nondeterministic domains can be time-consuming due to the many contingencies. However, real-time heuristic search methods allow agents to gather information early. This information can be used to resolve some of the uncertainty caused by nondeterminism and thus reduce the amount of planning done for unencountered situations. To this end, we introduce Min-Max Learning Real-Time A* (Min-Max LRTA*), a real-time heuristic search method that generalizes Korf's LRTA* to nondeterministic domains. Min-Max LRTA* has the following advantages: First, different from the many existing ad-hoc planning methods that interleave planning and plan executions, it has a solid theoretical foundation and is domain independent. Second, it allows for fine-grained control over how much planning to do between plan executions. Third, it can use heuristic knowledge to guide planning which can reduce planning time without increasing the plan-execution time. Fourth, it can be interrupted at any state and resume execution at a different state. Fifth, it amortizes learning over several planning episodes, which allows it to find a plan with a suboptimal plan-execution time fast and then improve the plan-execution time as it solves similar planning tasks, until the plan-execution time is satisficing or optimal. Thus, it still asymptotically minimize the plan-execution time in the long run in case similar planning tasks unexpectedly repeat.

We apply Min-Max LRTA* to simulated robot-navigation tasks in mazes, where the robots know the maze but do not know their initial position and orientation (pose). These planning tasks can be modeled as planning tasks in nondeterministic domains whose states are sets of poses. We show that Min-Max LRTA* solves the robot-navigation tasks fast, converges quickly, and requires only a small amount of memory. We also discuss how Min-Max LRTA* can be applied to moving-target search, the task of catching a moving target.

Planning in nondeterministic domains, including domains where the state of the world cannot be observed completely, is still poorly understood. The main contribution of this article is to provide insight into the behavior of planning methods in nondeterministic domains. However, our analytical results about the properties of Min-Max LRTA* also apply to LRTA* and provide new insight into the behavior of this popular real-time heuristic search method in deterministic domains.

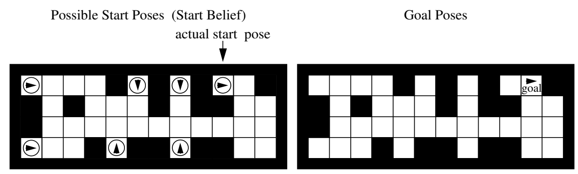


Figure 1: Example of a Goal-Directed Navigation Task

1 Introduction

Planning faces new challenges as planning methods get integrated into situated intelligent systems (agents). Classical planners traditionally assume that domains are deterministic, and the few planners that can operate in nondeterministic domains often assume that the state of the world is completely observable. Mobile robots and other agents, however, operate in nondeterministic domains, including domains where the state of the world cannot be observed completely. Since many of these planning tasks do not necessarily repeat, it is often infeasible to preprogram the agents with appropriate responses and it is essential that they solve planning tasks efficiently on-line. However, planning in nondeterministic domains can be time-consuming due to the many contingencies, and finding plans with minimal plan-execution time is often intractable.

One general principle that can reduce the planning time in nondeterministic domains is interleaving planning and plan executions [13]. A domain is nondeterministic if agents are not able to predict with certainty which successor states action executions result in. Without interleaving planning and plan executions, the agents have to find a large conditional plan that solves the planning task. When interleaving planning and plan executions, on the other hand, the agents have to find only the beginning of such a plan. After the execution of this subplan, the agents 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. Since actions are executed before their complete consequences are known, the agents are likely to incur some overhead in terms of the number of actions executed, but this is often outweighed by the computational savings gained. As an example, consider robot-navigation tasks in mazes where the robots know the maze but do not know their initial position and orientation (pose). Interleaving planning and plan executions allows the robots to make additional observations, which reduces their pose uncertainty and thus the number of situations that their plans have to cover. This makes subsequent planning more efficient. For example, assume that a robot has no knowledge of its start pose for the goal-directed navigation task from Figure 1, but observes walls around it except in its front. This results in the seven possible start poses shown in the figure. If the robot executes a forward action (the only sensible action given its observations) and then senses walls in its front and to its left and openings on the other two sides, it can infer that it has reached the goal pose and does not need to plan any further. In particular, it does not need to decide what it should have done had the observation been different.

We study how to interleave planning and plan executions in nondeterministic domains, and present both planning methods and first analytical results about their behavior. Planning methods that interleave planning and plan executions have to overcome two problems: first, they have to make sure that they make progress towards the goal instead of cycling forever; second, they should be able to improve their plan-execution time as they solve similar planning tasks, otherwise they do not behave efficiently in the long run in case similar planning tasks unexpectedly repeat. We show how Min-Max Learning Real-Time A* (Min-Max LRTA*), a real-time heuristic search method that extends LRTA* [32] to nondeterministic domains, can be used to address these problems. Min-Max LRTA* associates information with the states to prevent cycling, interleaves planning and plan executions, and plans only in the part of the domain around the current state of the agents (agent-centered search). This is the part of the domain that is immediately relevant for the agents in their current situation. Min-Max LRTA* has the following properties: First, different from the many existing ad-hoc planning methods that interleave planning and plan executions, Min-Max LRTA* has a solid theoretical foundation and is domain independent. Second, it allows for fine-grained control over how much planning to do between plan executions. Third, it can use heuristic knowledge to guide planning which can reduce planning time without increasing the plan-execution time. Fourth, it can be interrupted at any state and resume execution at a different state.

In other words, other control programs can take over control at arbitrary times if desired. Fifth, it amortizes learning over several planning episodes, which allows it to find a plan with a suboptimal plan-execution time fast and then improve the plan-execution time as it solves similar planning tasks, until the plan-execution time is satisficing or optimal. Thus, it still asymptotically minimizes the plan-execution time in the long run in case similar planning tasks unexpectedly repeat. As recognized in [11], this is an important property because no planning method that executes actions before it has found a complete plan can guarantee a good plan-execution time right away. Most planning methods that interleave planning and plan executions cannot improve their plan-execution time as they solve similar planning tasks and thus do not behave efficiently in the long run in case similar planning tasks unexpectedly repeat.

The algorithmic contribution of this article is the introduction of Min-Max LRTA*, a real-time heuristic search method for nondeterministic domains. The article also shows how the search space of Min-Max LRTA* can be represented more compactly than is possible with minimax trees and how it can be solved efficiently using methods from Markov game theory. Min-Max LRTA* generalizes LRTA* [32], probably the most popular real-time heuristic search method. Thus, our analytical results about the properties of Min-Max LRTA* also apply to this real-time heuristic search method that operates in deterministic domains. Min-Max LRTA* can also be changed so that it becomes very similar to Trial-Based Real-Time Dynamic Programming [1, 2] and thus also many reinforcement-learning methods. While our analytical results do not directly apply to this case, they still suggest properties of these dynamic programming methods.

To illustrate the advantages of Min-Max LRTA*, we apply it to simulated robot-navigation tasks in mazes, where the robots know the maze but do not know their initial pose. We show that Min-Max LRTA* solves these robot-navigation tasks fast, converges quickly, and requires only a small amount of memory. Min-Max LRTA* generalizes the Information-Gain Method that has been used to solve the robot-navigation tasks on actual robots but does not improve its plan-execution time as it solves similar planning tasks [13]. We also discuss how Min-Max LRTA* can be applied to moving-target search, the task of catching a moving target, thus providing an alternative to existing moving-target search methods.

However, the contributions of this article extend beyond Min-Max LRTA*. It contributes to the area of real-time heuristic search in three ways. First, our analysis demonstrates how to analyze methods that interleave planning and plan executions and thus contributes to a solid theoretical foundation of these methods. Second, the solid theoretical foundation of Min-Max LRTA* makes it a good stepping stone towards both a better understanding of planning in nondeterministic domains and more powerful and complex planning methods for these domains, including real-time heuristic search methods. Third, this article illustrates an advantage of real-time heuristic search methods that has not been studied in depth. Real-time heuristic search methods have mostly been studied in deterministic domains. For example, they are among the few search methods that are able to find suboptimal plans for the twenty-four puzzle, a sliding-tile puzzle with more than 7×10^{24} states [33]. They have also been used to solve large STRIPS-type planning tasks [7]. However, real-time heuristic search methods compete in deterministic domains with other suboptimal heuristic search methods such as greedy (best-first) search [46], that can find plans faster than LRTA*, or linear-space best-first search [45, 33], that can consume less memory [33, 5]. Many domains from robotics, control, and scheduling are nondeterministic, and real-time search methods allow agents to gather information early in nondeterministic domains. This information can be used to resolve some of the uncertainty caused by nondeterminism and thus reduce the amount of planning done for unencountered situations. Other contributions of this article include that it uses existing nondeterministic domains as example domains which provides additional insight into their properties and promotes their use as testbeds, and that it discusses relationships between various methods from artificial intelligence and related disciplines, including how these methods can be applied to planning tasks in nondeterministic domains and what their respective advantages and disadvantages are.

In the following, we first give an overview of LRTA*, an existing real-time heuristic search method that operates in deterministic domains. We then describe how it can be extended to nondeterministic domains, analyze the plan-execution time of the resulting real-time heuristic search method (Min-Max LRTA*), and apply it to robot-navigation and moving-target search tasks.

Initially, the non-negative u-values $u(s)$ are approximations of the goal distances (measured in action executions) for all $s \in S$.

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

Figure 2: Deterministic Domains: LRTA* with Look-Ahead One

2 Deterministic Domains: Learning Real-Time A*

In this section, we describe the *Learning Real-Time A* method* (LRTA*) [32]. We use the following notation to describe deterministic and nondeterministic planning tasks: S denotes the finite set of states of the domain, $s_{start} \in S$ the start state, and $G \subseteq S$ the set of goal states. The number of states is $n := |S|$. $A(s) \neq \emptyset$ is the finite, nonempty set of (potentially nondeterministic) actions that can be executed in state $s \in S$. $succ$ is the *transition function*, a function from states and actions to sets of states: $succ(s,a)$ denotes the set of successor states that can result from the execution of action $a \in A(s)$ in state $s \in S$. In deterministic domains, $succ(s,a)$ contains only one state and we use $succ(s,a)$ also to denote this state. An agent starts in the start state and has to move to a goal state. The agent always observes what its current state is and then has to select and execute its next action, which results in a state transition to one of the possible successor states. The planning task is solved when the agent reaches a goal state. We measure both the travel distance and plan-execution time of the agent in action executions, which is reasonable if every action has the same cost, for example, can be executed in about the same amount of time. (It is straightforward to extend both the real-time heuristic search methods in this article and the formal results about them to the case where the action costs are all different.) We also use two operators with the following semantics: The expression "$\arg\min_{x \in X} f(x)$" returns the elements $x \in X$ that minimize $f(x)$, that is, the set $\{x \in X : f(x) = \min_{x' \in X} f(x')\}$. Given such a set Y, the expression "one-of Y" returns an element of Y according to an arbitrary rule (that can, for example, include elements of chance). A subsequent invocation of "one-of Y" can return the same or a different element.

LRTA* associates a small amount of information with the states that allows it to remember where it has already searched. In particular, it associates a non-negative *u-value* $u(s)$ with each state $s \in S$. The u-values approximate the goal distances of the states. The u-values are updated as the search progresses and used to determine which actions to execute. Here, we describe LRTA* with look-ahead (search horizon) one (Figure 2). It consists of a *termination-checking step* (Line 2), an *action-selection step* (Line 3), a *value-update step* (Line 4), and an *action-execution step* (Line 5). LRTA* first checks whether it has already reached a goal state and thus can terminate successfully (Line 2). If not, it decides which action a to execute in the current state s (Line 3). It looks one action execution ahead and always greedily chooses an action that leads to a successor state with the minimal u-value (ties can be broken arbitrarily). If the u-value of this successor state plus one (for the action execution needed to reach the successor state) is larger than the u-value of the current state, then it replaces the u-value of the current state (Line 4). Finally, LRTA* executes the selected action (Line 5), updates the current state (Line 6), and iterates the procedure (Line 7). The planning time of LRTA* between action executions is linear in the number of actions available in the current state. If this number does not depend on the number of states, then the planning time between action executions does not depend on the number of states either.

The action-selection and value-update steps of LRTA* can be explained as follows. The action-selection step attempts to get to a goal state as fast as possible by always choosing an action that leads to a successor state with the minimal u-value. Since the u-values approximate the goal distances, this is a successor state that is believed to have the smallest goal distance. The value-update step uses the look-ahead of the action-selection step to make the u-value of the current state approximate the goal distance of the state better. The goal distance of a non-goal state is the minimum of the goal distances of its successor states plus one (for the action execution needed to reach the successor state). Consequently, $1+\min_{a \in A(s)} u(succ(s,a))$ approximates the goal distance of non-goal state s. If all u-values are admissible (that is, do not overestimate the goal distances), then both

Initially, the non-negative u-values $u(s)$ are approximations of the minimax goal distances (measured in action executions) for all $s \in S$.

1. $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. $a :=$ one-of $\arg\min_{a \in A(s)} \max_{s' \in succ(s,a)} u(s')$.
4. $u(s) := \max(u(s), 1 + \max_{s' \in succ(s,a)} u(s'))$.
5. Execute action a, that is, change the current state to a state in $succ(s,a)$ (according to the behavior of nature).
6. $s :=$ the current state.
7. Go to 2.

Figure 3: Nondeterministic Domains: Min-Max LRTA* with Look-Ahead One

Initially, the non-negative u-values $u(s)$ are approximations of the minimax goal distances (measured in action executions) for all $s \in S$.

1. $s := s_{start}$.
2. If $s \in G$, then stop successfully.
3. Generate a local search space S_{lss} with $s \in S_{lss}$ and $S_{lss} \cap G = \emptyset$.
4. Update $u(s)$ for all $s \in S_{lss}$ (Figure 5).
5. $a :=$ one-of $\arg\min_{a \in A(s)} \max_{s' \in succ(s,a)} u(s')$.
6. Execute action a, that is, change the current state to a state in $succ(s,a)$ (according to the behavior of nature).
7. $s :=$ the current state.
8. (If $s \in S_{lss}$, then go to 5.)
9. Go to 2.

Figure 4: Nondeterministic Domains: Min-Max LRTA*

$1 + \min_{a \in A(s)} u(succ(s,a))$ and $u(s)$ do not overestimate the goal distance of non-goal state s, and the larger of these two values is the more accurate estimate. For example, if the admissible u-values of the only two successor states of non-goal state s are one and three, then the length of a shortest path from state s via its first successor to a goal state is at least two and the length of a shortest path from state s via its second successor to a goal state is at least four. Since the shortest path from state s to a goal state has to go through one of its successors, the goal distance of state s is at least two. If the admissible u-value of state s is five, then its goal distance is at least five, the maximum of two and five. The value-update step then replaces the u-value of state s with this value. The value-update step of LRTA* is sometimes stated as $u(s) := 1 + u(succ(s,a))$. In this case, the value-update step replaces the u-value of state s from the above example with two, which decreases its u-value. Our slightly more complex version guarantees that the u-values are nondecreasing. Since the u-values remain admissible and larger admissible u-values tend to guide planning better than smaller admissible u-values, there is no reason to decrease them. If the u-values are consistent (that is, satisfy the triangle inequality) then both value-update steps are equivalent [25].

A more comprehensive introduction to LRTA* can be found in [20]. An example of the behavior of LRTA* is given in Section 4.3.

3 Nondeterministic Domains: Min-Max Learning Real-Time A*

In this section, we extend LRTA* to nondeterministic domains. A domain is nondeterministic if the agent is not able to predict with certainty which successor state an action execution results in. LRTA* and other real-time heuristic search methods have almost exclusively been applied to deterministic domains, such as sliding-tile puzzles [30, 31, 32, 47, 23, 33, 19], gridworlds [3, 32, 21, 18, 44, 54, 57, 37, 52, 19], blocksworlds [23, 7], and other STRIPS-type planning domains including domains from logistics [7]. However, many domains from robotics, control, and scheduling are nondeterministic. An example is robot-navigation in mazes, where the robots know the maze but do not know their initial position and orientation. We study these robot-navigation

The minimax-search method uses the temporary variables $u'(s)$ for all $s \in S_{lss}$.

1. For all $s \in S_{lss}$: $u'(s) := u(s)$ and $u(s) := \infty$.
2. If $u(s) < \infty$ for all $s \in S_{lss}$, then return.
3. $s' :=$ one-of $\arg\min_{s \in S_{lss}: u(s)=\infty} \max(u'(s), 1 + \min_{a \in A(s)} \max_{s'' \in succ(s,a)} u(s''))$.
4. If $\max(u'(s'), 1 + \min_{a \in A(s')} \max_{s'' \in succ(s',a)} u(s'')) = \infty$, then return.
5. $u(s') := \max(u'(s'), 1 + \min_{a \in A(s')} \max_{s'' \in succ(s',a)} u(s''))$.
6. Go to 2.

Figure 5: Minimax-Search Method

tasks in Section 6.

The *Min-Max Learning Real-Time A* Method* (Min-Max LRTA*) [25] uses minimax search to extend LRTA* to nondeterministic domains. Thus, similar to game-playing approaches and reinforcement-learning methods such as Q̂-Learning [16] or MARTDP [53], Min-Max LRTA* views acting in nondeterministic domains as a two-player game in which it selects an action from the available actions in the current state. This action determines the possible successor states from which a fictitious agent, called *nature*, chooses one. Min-Max LRTA* assumes that nature exhibits the most vicious behavior and always chooses the worst possible successor state, an assumption not only made in the context of planning in artificial intelligence [10, 14] but also manipulation and assembly planning in robotics [35]. Thus, Min-Max LRTA* attempts to move with every action execution as close to the goal state as possible under the assumption that nature is an opponent that tries to move it as far away from the goal state as possible. Acting in deterministic domains is then simply a special case where every action uniquely determines the successor state.

Like LRTA*, Min-Max LRTA* associates a small amount of information with the states that allows it to remember where it has already searched. In particular, it associates a non-negative *u-value* $u(s)$ with each state $s \in S$. The u-values approximate the minimax goal distances of the states. The u-values are updated as the search progresses and are used to determine which actions to execute. The *minimax goal distance* $gd(s) \in [0, \infty]$ of state $s \in S$ is the smallest number of action executions with which a goal state can be reached from state s, even for the most vicious behavior of nature. The minimax goal distances are defined by the following set of equations for all $s \in S$:

$$gd(s) = \begin{cases} 0 & \text{if } s \in G \\ 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} gd(s') & \text{otherwise} \end{cases} \qquad \text{for all } s \in S.$$

In deterministic domains, this definition is identical to the traditional definition of goal distances.

Thus, the difference between LRTA* with look-ahead one (Figure 2) and Min-Max LRTA* with look-ahead one (Figure 3) is the following: LRTA* operates in deterministic domains. Its u-values approximate the goal distances of the states and it uses $u(succ(s, a))$ in the action-selection and value-update steps. This expression is the u-value of the state that results from the execution of action a in state s. Min-Max LRTA*, on the other hand, operates in nondeterministic domains. Its u-values approximate the minimax goal distances of the states and it uses $\max_{s' \in succ(s,a)} u(s')$ in the action-selection and value-update steps. This expression is the worst u-value of all states that can result from the execution of action a in state s. To extend it to the case where the action costs are all different, the action-selection step becomes $a :=$ one-of $\arg\min_{a \in A(s)}(c(s, a) + \max_{s' \in succ(s,a)} u(s'))$, and the value-update step becomes $u(s) := \max(u(s), c(s, a) + \max_{s' \in succ(s,a)} u(s'))$, where $c(s, a)$ is the cost of executing action a in state s. In deterministic domains, Min-Max LRTA* and LRTA* behave identically.

So far, we have discussed only real-time heuristic search methods with look-ahead one, because these are the simplest and thus most easy-to-understand real-time heuristic search methods. However, larger look-aheads are important for slowly acting agents. In the following, we therefore generalize Min-Max LRTA* to arbitrary look-aheads, extending a previous approach [47] to nondeterministic domains by interleaving minimax searches in local search spaces and plan executions.

Our generalization of Min-Max LRTA* to arbitrary look-aheads is shown in Figure 4. It consists of a *termination-checking step* (Line 2), a *local-search-space-generation* step (Line 3), a *value-update step* (Line 4) that implements the minimax search, an *action-selection step* (Line 5), and an *action-execution step* (Line 6). Min-Max LRTA* first checks whether it has already reached a goal state and thus can terminate successfully (Line 2). If not, it generates the local search space $S_{lss} \subseteq S$ (Line 3). The states in the local search space correspond to the non-leaf nodes of the corresponding search tree, and thus are all non-goal states. While we require only that $s \in S_{lss}$ and $S_{lss} \cap G = \emptyset$, in practice Min-Max LRTA* constructs S_{lss} by searching forward from s. Min-Max LRTA* then uses its minimax-search method to update the u-values of all states in the local search space (Line 4). Based on these u-values, Min-Max LRTA* decides which action to execute next (Line 5). Finally, it executes the selected action (Line 6), updates its current state (Line 7), and iterates the procedure.

The minimax-search method that Min-Max LRTA* uses to update the u-values in the local search space is shown in Figure 5. It assigns each state its minimax goal distance under the assumption that the u-values of all states in the local search space do not overestimate the correct minimax goal distances and the u-values of all states outside of the local search space correspond to their correct minimax goal distances. It does this by first assigning infinity to the u-values of all states in the local search space. It then determines the state in the local search space whose u-value is still infinity and which minimizes the maximum of its previous u-value and the minimum of the current u-values of its successor states plus one (this formula corresponds exactly to the value-update step of Min-Max LRTA* with look-ahead one). The u-value of this state is then assigned this value, and the process repeats. The way the u-values are updated ensures that the states in the local search space are updated in the order of their increasing new u-values. This ensures that the u-value of each state in the local search space is updated at most once. The method terminates when either the u-value of each state in the local search space has been assigned a finite value or a u-value would be assigned the value infinity. In the latter case, the u-values of all remaining states in the local search space would be assigned the value infinity as well, which is already their current value. The following theorem formalizes the statement that the resulting u-values correspond to the minimax goal distances of the states under the assumption that the u-values of all states in the local search space do not overestimate the correct minimax goal distances and the u-values of all states outside of the local search space correspond to their correct minimax goal distances.

Theorem 1 *For all times $t = 0, 1, 2, \ldots$ (until termination): Consider the $(t+1)$st value-update step (minimax search) of Min-Max LRTA* (Line 4 in Figure 4). Let $u^t(s) \in [0, \infty]$ and $u^{t+1}(s) \in [0, \infty]$ refer to the u-values immediately before and after, respectively, the minimax search. Then, the minimax search terminates with*

$$u^{t+1}(s) = \begin{cases} u^t(s) & \text{if } s \notin S^t_{lss} \\ \max(u^t(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} u^{t+1}(s')) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S.$$

Proof: See the appendix. ■

Min-Max LRTA* could represent the local search spaces as minimax trees, which could be searched with traditional minimax-search methods. However, this has the disadvantage that the memory requirements and the search effort can be exponential in the depth of the tree (the look-ahead of Min-Max LRTA*) although the number of different states grows often only polynomially in the depth of the trees.[1] To take advantage of this property, Min-Max LRTA* represents the local search spaces more compactly as and-or graphs that contain every state at most once. This requires a more sophisticated minimax-search method because there can now be paths of different lengths between any two states in the graph. Min-Max LRTA* uses a minimax-search method that is related to more general (and often more efficient) dynamic programming methods from Markov game theory [34], that could also be used but are harder to prove correct.

[1] As an example, assume that a robot solves the robot-navigation tasks from Section 6 in a large empty space where it cannot observe any walls within its look-ahead. In this case, a search tree of depth d contains $3^0 + 3^1 + \ldots 3^d = 3^{d+1}/2 - 1/2$ nodes since the robot can always execute three actions. The number of beliefs that can be reached from the current belief with at most d forward actions (and an arbitrary number of turn actions) is $8d^2 + 8d + 4$, which is an upper bound on the number of different beliefs in a search tree of depth d. Thus, the number of nodes in the search tree grows exponentially in its depth but the number of different states grows only polynomially.

After the minimax-search method has updated the u-values, Min-Max LRTA* greedily chooses the action for execution that minimizes the u-value of the successor state in the worst case (ties can be broken arbitrarily). The rationale behind this action-selection step is that the u-values approximate the minimax goal distances and Min-Max LRTA* attempts to decrease its minimax goal distance as much as possible. Then, Min-Max LRTA* has a choice. It can generate another local search space, update the u-values of all states that it contains, and select another action for execution. If the new state is still part of the local search space (the one that was used to determine the action whose execution resulted in the new state), Min-Max LRTA* can also select another action for execution based on the current u-values (Line 8). We analyze Min-Max LRTA* without Line 8 but use Min-Max LRTA* with Line 8 in the examples and experiments because Min-Max LRTA* with Line 8 utilizes more information of the minimax searches in the local search spaces. We can proceed this way because Min-Max LRTA* with Line 8 is a special case of Min-Max LRTA* without Line 8: After Min-Max LRTA* has run the minimax-search method on some local search space, the u-values do not change if Min-Max LRTA* runs the minimax-search method again on the same local search space or a subset thereof. Whenever Min-Max LRTA* with Line 8 jumps to Line 5, the new current state is still part of the local search space S_{lss} and thus not a goal state. Consequently, Min-Max LRTA* can skip Line 2. Min-Max LRTA* could now search a subset of S_{lss} that includes the new current state s, for example S_{lss} (again) or $\{s\}$. Since this does not change the u-values, Min-Max LRTA* can, in this case, also skip the minimax search. Section 6.4 presents several complete example runs of Min-Max LRTA* with local search spaces of different sizes in the context of robot-navigation tasks, and also steps through the minimax-search method.

Min-Max LRTA* with local search space $S_{lss} = \{s\}$ (Figure 4) and Min-Max LRTA* with look-ahead one (Figure 3) behave identically in domains with the following property: the execution of all actions in non-goal states necessarily results in a state change, that is, it cannot leave the current state unchanged. In general, these actions can safely be deleted from the domains because there always exists a minimax solution that does not use them if there exists a minimax solution at all. For example, the optimal minimax solution does not use them. Min-Max LRTA* with any local search space, including $S_{lss} = \{s\}$, never executes actions whose execution can leave the current state unchanged but Min-Max LRTA* with look-ahead one can execute them. In the following, we refer to Figure 4 and not Figure 3 when we analyze the plan-execution time of Min-Max LRTA*. [25] explains how the presence of actions whose execution can leave the current state unchanged affects the plan-execution time of Min-Max LRTA* with look-ahead one.

4 Analysis of the Plan-Execution Time of Min-Max LRTA*

Min-Max LRTA* is similar to game-playing programs that use minimax search, except that Min-Max LRTA* modifies its evaluation function during the search. This is interesting since LRTA* was originally inspired by game-playing programs. Both game-playing programs and Min-Max LRTA* have to search large nondeterministic domains where the behavior of the opponent is unknown. Both make the planning task tractable by interleaving minimax searches and plan executions. However, their planning objectives differ. Game-playing programs distinguish terminal states of different quality (wins, losses, and draws). Their objective is to get to a winning terminal state. How long this takes is usually unimportant. Min-Max LRTA*, on the other hand, has only winning terminal states (the goal states). Its objective is to get to a terminal state fast. In this section, we therefore analyze how quickly Min-Max LRTA* reaches a goal state.

The *performance* of Min-Max LRTA* is its plan-execution time, measured in the number of actions that it executes until it reaches a goal state. This is motivated by the fact that, for sufficiently fast acting agents, the time until a problem is solved is determined by the planning time, which is roughly proportional to the number of action executions if Min-Max LRTA* performs only a constant amount of computations between action executions. (Thus, the performance measure cannot be used to choose how much planning to do between plan executions but it can be used to compare two real-time heuristic search methods that both perform about the same amount of planning between plan executions.) For sufficiently slowly acting agents, on the other hand, the time until a problem is solved is determined by the plan-execution time, which is roughly proportional to the number of action executions if every action can be executed in about the same amount of time. The *complexity* of Min-Max LRTA* is its worst-case performance over all possible topologies of domains with the same number of states, all possible start and goal states, all tie-breaking rules among indistinguishable actions, and all strategies of nature.

Experimental researchers sometimes consider a complexity analysis of search methods to be unimportant because they are more interested in their average performance than their worst-case performance. The reason why a complexity analysis is interesting, however, is because it provides performance guarantees in form of upper bounds on the performance of search methods. We therefore perform a complexity analysis of Min-Max LRTA*.

4.1 Assumptions

In this section, we describe the assumptions that underlie our complexity analysis of Min-Max LRTA*.

One advantage of Min-Max LRTA* is that it does not depend on assumptions about the behavior of nature. This is so because minimax searches assume that nature is vicious and always chooses the worst possible successor state. If Min-Max LRTA* can reach a goal state for the most vicious behavior of nature, it also reaches a goal state if nature uses a different and therefore less vicious behavior. A disadvantage of Min-Max LRTA* is that it cannot solve all planning tasks. This is so because it interleaves minimax searches and plan executions. Minimax searches limit the solvable planning tasks because they are overly pessimistic. They can solve only planning tasks for which the minimax goal distance of the start state is finite. Interleaving planning and plan execution limits the solvable planning tasks further because it executes actions before their complete consequences are known. Thus, even if the minimax goal distance of the start state is finite, it is possible that Min-Max LRTA* accidentally executes actions that lead to a state whose minimax goal distance is infinite, at which point the planning task might have become unsolvable. We discuss later how these problems can be addressed. Despite these problems, Min-Max LRTA* is guaranteed to solve all safely explorable domains. A domain is *safely explorable* if and only if the minimax goal distances of all states are finite. To be precise: First, all states of the domain can be deleted that cannot possibly be reached from the start state or can be reached from the start state only by passing through a goal state. The minimax goal distances of all remaining states have to be finite. Safely explorable domains guarantee that Min-Max LRTA* is able to reach a goal state no matter which actions it has executed in the past and what the behavior of nature is. We discuss an example of a safely explorable domain in Section 6. We assume for now that Min-Max LRTA* is applied to safely explorable domains but discuss later how this assumption can be relaxed.

4.2 Properties of U-Values

In this section, we define the properties of u-values needed for the complexity analysis.

Definition 1 *U-values are* uniformly initialized *with x (or, synonymously, x-initialized) if and only if initially*

$$u(s) = \begin{cases} 0 & \text{if } s \in G \\ x & \text{otherwise} \end{cases} \qquad \text{for all } s \in S.$$

Definition 2 *U-values are* admissible *if and only if*

$$0 \le u(s) \le gd(s) \qquad \text{for all } s \in S.$$

Admissibility means that the u-values do not overestimate the minimax goal distances. In deterministic domains, this definition reduces to the traditional definition of admissible heuristic values for A* search [43]. Zero-initialized u-values, for example, are admissible.

Admissible u-values have the following important property. This is the same property that we motivated in Section 2 for LRTA*.

Theorem 2 *Admissible initial u-values remain admissible after every value-update step of Min-Max LRTA* and are monotonically nondecreasing.*

Proof: See the appendix. ■

This theorem generalizes a theorem in [21], that they state for admissible initial u-values of LRTA* in deterministic domains. Their theorem is about the Moving-Target Search method (MTS) [21], that we discuss in Section 7, but MTS behaves identically to LRTA* if the target does not move.

4.3 Upper Bound on the Plan-Execution Time of Min-Max LRTA*

In this section, we provide an upper bound on the complexity of Min-Max LRTA* without Line 8. Our analysis is centered around the invariant from Theorem 3. The time superscript t refers to the values of the variables immediately before the $(t+1)$st value-update step of Min-Max LRTA* (Line 4 in Figure 4). For example, s^t denotes the state before the $(t+1)$st value-update step. For instance, $s^0 = s_{start}$. Similarly, $u^t(s)$ denotes the u-values before the $(t+1)$st value-update step and $u^{t+1}(s)$ the u-values after the $(t+1)$st value-update step. In the following, we prove an upper bound on the number of action executions after which Min-Max LRTA* is guaranteed to reach a goal state in safely explorable domains. The idea is simple. According to Theorem 2, the value-update step cannot decrease the u-values. After the value-update step it holds that $u(s^t) \geq 1 + u(s^{t+1})$ since the current state s^t is part of the local search space at time t. From time t to $t+1$, Min-Max LRTA* changes the current state from s^t to s^{t+1} and increases the value of the potential $\sum_{s \in S \setminus \{s^t\}} u^t(s)$ (the sum of the u-values of all states but the current state) by $u(s^t) - u(s^{t+1}) \geq 1$, that is, by at least one with every action execution. The u-value of every state is always bounded from above by the minimax goal distance of the state since the u-values are initially admissible and remain admissible after every action execution according to Theorem 2. Consequently, the value of the potential $\sum_{s \in S \setminus \{s^t\}} u^t(s)$ is bounded from above by $\sum_{s \in S} gd(s)$ at all times t. The value of the potential is $\sum_{s \in S \setminus \{s^0\}} u^0(s)$ at time $t = 0$. It increases by at least one with every action execution and is bounded from above by $\sum_{s \in S} gd(s)$. Consequently, the number of action executions is bounded from above by $\sum_{s \in S} gd(s) - \sum_{s \in S \setminus \{s^0\}} u^0(s) = u^0(s_{start}) + \sum_{s \in S} [gd(s) - u^0(s)]$. In the following, we formalize the proof. Since Min-Max LRTA* without Line 8 executes one action after each value-update step, the invariant from Theorem 3 shows that the number of executed actions t is always bounded from above by an expression that depends only on the initial and current u-values.

Theorem 3 *For all times* $t = 0, 1, 2, \ldots$ *(until termination)*

$$t \quad \leq \quad \sum_{s \in S} \left[u^t(s) - u^0(s)\right] - (u^t(s^t) - u^0(s^0))$$

for Min-Max LRTA with admissible initial u-values in safely explorable domains, regardless of the behavior of nature.*[2]

Proof by induction: The u-values are admissible at time t according to Theorem 2. Thus, they are bounded from above by the minimax goal distances, which are finite since the domain is safely explorable. For $t = 0$, the inequality reduces to $t \leq 0$, which is true. Now assume that the theorem holds at time t. The left-hand side of the inequality increases by one between time t and $t+1$. The right-hand side of the inequality increases by

$$\begin{aligned} & \sum_{s \in S \setminus \{s^{t+1}\}} u^{t+1}(s) - \sum_{s \in S \setminus \{s^t\}} u^t(s) \\ = & \sum_{s \in S \setminus \{s^t, s^{t+1}\}} [u^{t+1}(s) - u^t(s)] + u^{t+1}(s^t) - u^t(s^{t+1}) \end{aligned}$$

[2] Sums have a higher precedence than other operators. For example, $\sum_i x + y = \sum_i [x] + y \neq \sum_i [x+y]$.

$$
\begin{aligned}
&\overset{\text{Theorem 1}}{=} \sum_{s \in S \setminus \{s^t, s^{t+1}\}} [u^{t+1}(s) - u^t(s)] + \max(u^t(s^t), 1 + \min_{a \in A(s^t)} \max_{s' \in succ(s^t,a)} u^{t+1}(s')) - u^t(s^{t+1}) \\
&\geq \sum_{s \in S \setminus \{s^t, s^{t+1}\}} [u^{t+1}(s) - u^t(s)] + \min_{a \in A(s^t)} \max_{s' \in succ(s^t,a)} u^{t+1}(s') - u^t(s^{t+1}) + 1 \\
&\geq \sum_{s \in S \setminus \{s^t, s^{t+1}\}} [u^{t+1}(s) - u^t(s)] + u^{t+1}(s^{t+1}) - u^t(s^{t+1}) + 1 \\
&= \sum_{s \in S \setminus \{s^t\}} [u^{t+1}(s) - u^t(s)] + 1 \\
&\overset{\text{Theorem 2}}{\geq} 1. \blacksquare
\end{aligned}
$$

Theorem 4 uses Theorem 3 to derive an upper bound on the number of action executions.

Theorem 4 *Let $u^0(s)$ denote the initial u-values. Then, Min-Max LRTA* with admissible initial u-values reaches a goal state after at most $u^0(s_{start}) + \sum_{s \in S}[gd(s) - u^0(s)]$ action executions in safely explorable domains, regardless of the behavior of nature.*

Proof:

$$
\begin{aligned}
t &\overset{\text{Theorem 3}}{\leq} \sum_{s \in S} \left[u^t(s) - u^0(s)\right] - (u^t(s^t) - u^0(s^0)) \\
&\overset{\text{Admissibility}}{\leq} \sum_{s \in S} \left[gd(s) - u^0(s)\right] + u^0(s^0). \\
&= u^0(s_{start}) + \sum_{s \in S}[gd(s) - u^0(s)] \ \blacksquare
\end{aligned}
$$

Since the minimax goal distances are finite in safely explorable domains, Theorem 4 shows that Min-Max LRTA* with admissible initial u-values reaches a goal state after a bounded number of action executions in safely explorable domains, that is, it is correct. More precisely: Min-Max LRTA* reaches a goal state after at most $\sum_{s \in S} gd(s)$ action executions, no matter what its heuristic knowledge is. One consequence of this result is that state spaces where all states are clustered around the goal states are easier to solve with Min-Max LRTA* than state spaces that do not possess this property. In domains with this property, Min-Max LRTA* always remains in the vicinity of a goal state even though it executes suboptimal actions from time to time. It holds that $\sum_{s \in S} gd(s) \leq \sum_{i=0}^{n-1} i = n^2/2 - n/2$. Now assume that Min-Max LRTA* with local search space $S_{lss} = \{s\}$ is zero-initialized, which implies that it is uninformed. In the following, we show that the upper complexity bound is then tight for infinitely many n. Our example domains are deterministic. This shows that the upper complexity bound of Min-Max LRTA* is tight for this important subclass of nondeterministic domains and that deterministic domains, in general, are not easier to search with Min-Max LRTA* than nondeterministic domains. It also shows that the upper complexity bound is tight for zero-initialized LRTA* with local search space $S_{lss} = \{s\}$ since, in deterministic domains, Min-Max LRTA* and LRTA* behave identically. Thus, the following example domains also provide insight into the behavior of LRTA*.

Figure 6 shows an example of a safely explorable domain for which the number of action executions that zero-initialized Min-Max LRTA* with local search space $S_{lss} = \{s\}$ needs in the worst case to reach a goal state is $n^2/2 - n/2$. The upper part of the figure shows the state space. The states are annotated with their minimax goal distances, their initial heuristic values, and their names. The lower part of the figure shows the behavior of Min-Max LRTA*. On the right, the figure shows the state space with the u-values after the value-update step but before the action-execution step. The current state is shown in bold. On the left, the figure shows the minimax searches that resulted in the u-values shown on the right. Again, the states are annotated with their u-values after the value-update step but before the action-execution step. The current state is at the top. Ellipses

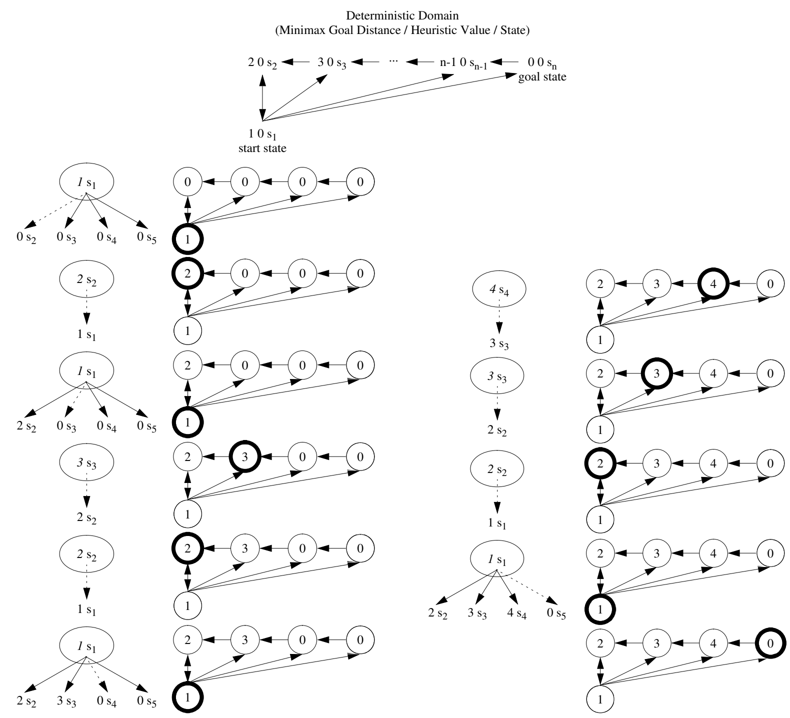


Figure 6: Worst-Case Domain for Min-Max LRTA*

show the local search spaces, and dashed lines show the actions that Min-Max LRTA* is about to execute. For the example domain, after Min-Max LRTA* has visited a state for the first time, it has to move through all previously visited states again before it is able visit another state for the first time. Thus, the number of action executions is quadratic in the number of states. We provide pseudo-code that prints the exact sequence of states that Min-Max LRTA* traverses. The scope of the for-statements is shown by indentation. The statements in their scope get executed only if the range of the for-variable is not empty. The default step size of the for-statements is either one ("to") or minus one ("downto"). It is easy to determine from an inspection of the code how many actions Min-Max LRTA* executes. Min-Max LRTA* traverses the state sequence that is printed by the following program in pseudo code if ties are broken in favor of successor states with smaller indices.

```
for i := 1 to n-1
  for j := i downto 1
    print j
print n
```

In this case, Min-Max LRTA* executes $n^2/2 - n/2$ actions before it reaches the goal state (for $n \geq 1$). The number of executed actions exactly equals the sum of the goal distances because Min-Max LRTA* was zero-initialized and its final u-values are equal to the minimax goal distances. For example, $n^2/2 - n/2 = 10$ for $n = 5$. In this case, Min-Max LRTA* traverses the state sequence s_1, s_2, s_1, s_3, s_2, s_1, s_4, s_3, s_2, s_1, and s_5 of length ten. Figure 6 visualizes this execution trace.

The domain from Figure 6 was artificially constructed. However, Figure 7 shows a safely explorable gridworld for which the number of action executions that zero-initialized Min-Max LRTA* with local search space $S_{lss} = \{s\}$ needs in the worst case to reach a goal state is still on the order of n^2. This domain is undirected and the number of actions that can be executed in any state is bounded from above by a small constant (here: three). The figure also shows the behavior of Min-Max LRTA*. Min-Max LRTA* can "paint itself into a corner" repeatedly by moving around until suddenly the u-values of all successor states have increased by one and it thus is on another plateau of the value surface without knowing where the goal state is, resulting in a number of action executions that is quadratic in the number of states. Min-Max LRTA* can traverse the state sequence that is printed by the following program in pseudo code.

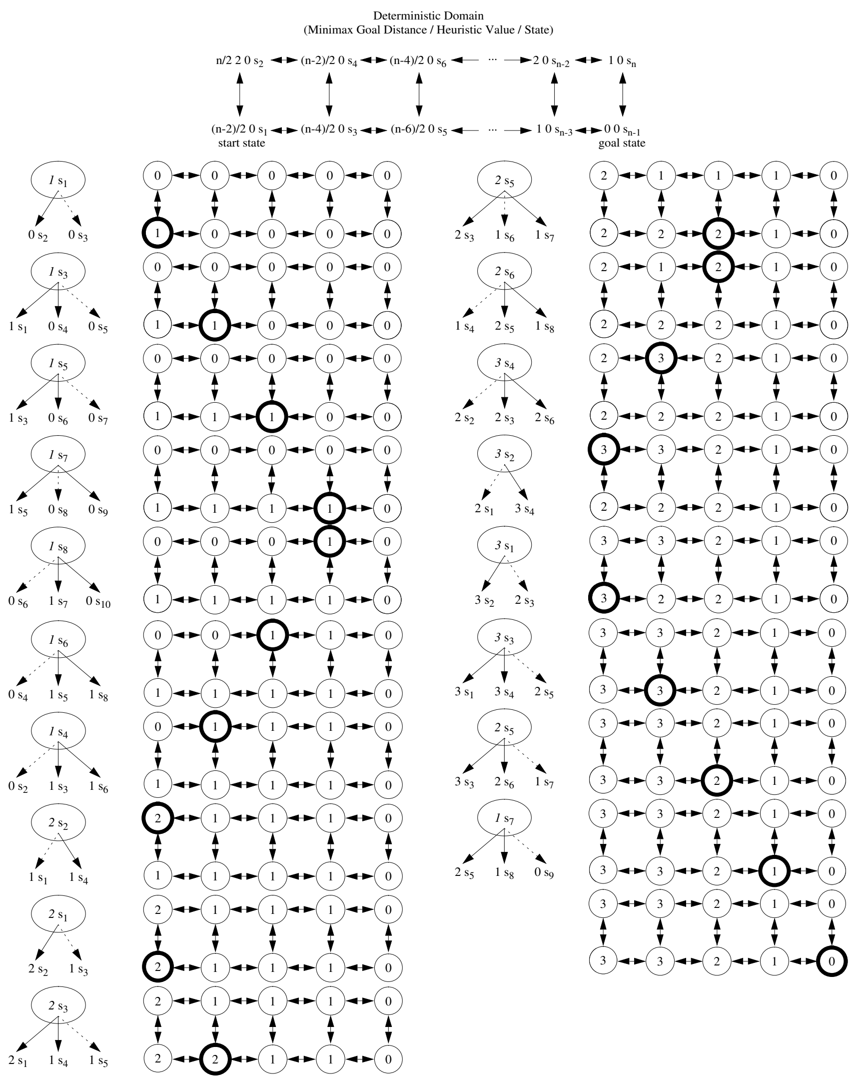


Figure 7: Rectangular Gridworld

```
for i := n-3 downto n/2 step 2
  for j := 1 to i step 2
    print j
  for j := i+1 downto 2 step 2
    print j
for i := 1 to n-1 step 2
  print i
```

In this case, Min-Max LRTA* executes $3n^2/16 - 3/4$ actions before it reaches the goal state (for $n \geq 2$ with $n \bmod 4 = 2$). The number of executed actions is bounded from above by but not equal to the sum of the goal distances because the final u-values are bounded from above by the goal distances but are not equal to them in all cases. For example, $3n^2/16 - 3/4 = 18$ for $n = 10$. In this case, Min-Max LRTA* traverses the state sequence s_1, s_3, s_5, s_7, s_8, s_6, s_4, s_2, s_1, s_3, s_5, s_6, s_4, s_2, s_1, s_3, s_5, s_7, and s_9. Figure 7 visualizes this execution trace.

5 Features of Min-Max LRTA*

Min-Max LRTA* has three key features, namely that it uses heuristic knowledge to guide planning, allows for fine-grained control over how much planning to do between plan executions, and improves its plan-execution time as it solves similar planning tasks, until its plan-execution time is at least worst-case optimal. The third feature is especially important since no search method that executes actions before it knows their complete consequences can guarantee a good plan-execution time right away. In this section, we explain these features of Min-Max LRTA*.

5.1 Heuristic Knowledge

Min-Max LRTA* uses heuristic knowledge to guide planning. This knowledge is provided in the form of admissible initial u-values. The larger its initial u-values, the better informed Min-Max LRTA* and the smaller the upper bound on the number of action executions provided by Theorem 4. For example, Min-Max LRTA* is fully informed if its initial u-values equal the minimax goal distances of the states. In this case, Theorem 4 predicts that Min-Max LRTA* reaches a goal state after at most $gd(s_{start})$ action executions. Thus, its plan-execution time is at least worst-case optimal and no other search method can do better in the worst-case.

Admissible heuristic functions are known for many deterministic domains. For nondeterministic domains, admissible u-values can be obtained as follows: One can assume that nature decides in advance which successor state $g(s, a) \in succ(s, a)$ to choose whenever action $a \in A(s)$ is executed in state $s \in S$; all possible states are fine. If nature really behaved this way, then the domain would effectively be deterministic. U-values that are admissible for this deterministic domain are admissible for the nondeterministic domain as well, regardless of the actual behavior of nature. This is so because additional action outcomes allow a vicious nature to cause more harm: U-values that are admissible in the deterministic domain are also admissible in the nondeterministic domain because, for each state, its minimax goal distance in the nondeterministic domain is at least as large as its goal distance in the deterministic domain. How informed the obtained u-values in the nondeterministic domain are depends on how informed they are in the deterministic domain and how close the assumed behavior of nature is to its most vicious behavior.

5.2 Fine-Grained Control

Min-Max LRTA* allows for fine-grained control over how much planning to do between plan executions. Thus, it is an any-time contract algorithm [48]. For example, Min-Max LRTA* with line 8 and $S_{lss} = S \setminus G = S \cap \overline{G}$ (or, in general, sufficiently large local search spaces) performs a complete minimax search without interleaving planning and plan executions, which is slow but produces plans whose plan-execution times are worst-case optimal. On the other hand, Min-Max LRTA* with $S_{lss} = \{s\}$ performs almost no planning between plan executions. Smaller local search spaces benefit agents that can execute plans with a similar speed as they can generate them, because they tend to reduce the planning time. For example, when real-time heuristic search methods are used to solve deterministic planning tasks off-line, they only move a marker within the computer (that represents the state of the fictitious agent) and thus plan execution is fast. Korf [32] studies which look-ahead minimizes the planning time of the Real-Time A* method (RTA*) [30], a variant of LRTA*, in this context. He reports that a look-ahead of one is optimal for the eight puzzle and a look-ahead of two is optimal for the fifteen and twenty-four puzzles if the Manhattan distance is used to initialize the u-values. Knight [23] reports similar results. Larger local search spaces are needed for more slowly acting agents. It might be possible for Min-Max LRTA* to determine good local search spaces automatically by using techniques developed for real-time heuristic search methods in deterministic domains [18] or general techniques from limited rationality and deliberation scheduling [4, 17, 58], an application of which is described in [47].

5.3 Improvement of Plan-Execution Time

If Min-Max LRTA* solves the same planning task repeatedly (even with different start states) it can improve its plan-execution time by transferring domain knowledge, in the form of u-values, between planning tasks. This is a familiar concept since one can argue that the minimax searches that Min-Max LRTA* performs between plan executions are independent of one another and are connected only via the u-values that transfer domain knowledge between them.

Assume that a series of planning tasks in the same domain with the same set of goal states are given. The start states need not be identical. If the initial u-values of Min-Max LRTA* are admissible for the first planning task, then they are also admissible for the first planning task after Min-Max LRTA* has solved the task and are state-wise at least as informed as initially. Thus, they are also admissible for the second planning task and Min-Max LRTA* can continue to use the same u-values across planning tasks. The start states of the planning tasks can differ since the admissibility of u-values does not depend on them. This way, Min-Max LRTA* can transfer acquired domain knowledge from one planning task to the next, thereby making its u-values better informed. Ultimately, better informed u-values result in an improved plan-execution time, although the improvement is not necessarily monotonic. (This also explains why Min-Max LRTA* can be interrupted at any state and resume execution at a different state.) The following theorems formalize this knowledge transfer in the mistake-bounded error model. The mistake-bounded error model is one way of analyzing learning methods by bounding the number of mistakes that they make. We first prove that Min-Max LRTA* reaches a goal state after at most $gd(s_{start})$ action executions in safely explorable domains if its u-values do not change during the search. The idea is simple. Since the u-values do not change, we do not need to index them with the time t. After the value-update step it holds that $u(s^t) \geq 1 + u(s^{t+1})$ since the current state s^t is part of the local search space. From time t to $t+1$, Min-Max LRTA* changes the current state from s^t to s^{t+1} and decreases the u-value of the current state by $u(s^t) - u(s^{t+1}) \geq 1$, that is, by at least one with every action execution. The u-value of the current state is bounded from below by zero at all times t. The u-value of the current state is bounded from above by the minimax goal distance of the start state s_{start} at time $t = 0$ since the initial u-values are admissible. It decreases by at least one with every action execution and is bounded from below by zero. Consequently, the number of action execution is bounded from above by $gd(s_{start})$. In the following, we formalize the proof.

Theorem 5 *Min-Max LRTA* with admissible initial u-values reaches a goal state after at most $gd(s_{start})$ action executions in safely explorable domains, regardless of the behavior of nature, if its u-values do not change during the search.*

Proof:

$$
\begin{aligned}
u^t(s^t) - u^{t+1}(s^{t+1}) &= u^{t+1}(s^t) - u^{t+1}(s^{t+1}) && \text{since the u-values do not change} \\
&\overset{\text{Theorem 1}}{=} \max(u^t(s^t), 1 + \min_{a \in A(s^t)} \max_{s' \in succ(s^t,a)} u^{t+1}(s')) - u^{t+1}(s^{t+1}) \\
&\geq \min_{a \in A(s^t)} \max_{s' \in succ(s^t,a)} u^{t+1}(s') - u^{t+1}(s^{t+1}) + 1 \\
&\geq u^{t+1}(s^{t+1}) - u^{t+1}(s^{t+1}) + 1 \\
&= 1.
\end{aligned}
$$

Thus, the difference in u-values between the previous state and the current state is at least one. Since the u-values of all goal states are zero and the u-value of the start state is at most $gd(s_{start})$, by induction Min-Max LRTA* needs at most $gd(s_{start})$ action executions to reach a goal state. ■

Theorem 6 *Assume that Min-Max LRTA* maintains u-values across a series of planning tasks in the same safely explorable domain with the same set of goal states. Then, the number of planning tasks for which Min-Max LRTA* with admissible initial u-values reaches a goal state after more than $gd(s_{start})$ action executions* *(where s_{start} is the start state of the current planning task) is bounded from above by a finite constant that depends only on the domain and goal states.*

Proof: If Min-Max LRTA* with admissible initial u-values reaches a goal state after more than $gd(s_{start})$ action executions, then at least one u-value has changed according to Theorem 5. This can happen only a bounded number of times since the u-values are monotonically nondecreasing and remain admissible according to Theorem 2, and thus are bounded from above by the minimax goal distances, which are finite in safely explorable domains and depend only on the domain and goal states. ■

In this context, it counts as one mistake when Min-Max LRTA* reaches a goal state after more than $gd(s_{start})$ action executions. According to Theorem 6, the u-values converge after a bounded number of mistakes. The action sequence after convergence depends on the behavior of nature and is not necessarily uniquely determined, but has $gd(s_{start})$ *or fewer* actions, that is, the plan-execution time of Min-Max LRTA* is either worst-case optimal or *better than worst-case optimal*. This is possible because nature might not be as malicious as a minimax search assumes. Min-Max LRTA* might not be able to detect this "problem" by introspection since it does not perform a complete minimax search but partially relies on observing the actual successor states of action executions, and nature can wait an arbitrarily long time to reveal the "problem" or choose not to reveal it at all. This can prevent the u-values from converging after a bounded number of action executions (or planning tasks) and is the reason why we analyzed the behavior of Min-Max LRTA* using the mistake-bounded error model. It is important to realize that, since Min-Max LRTA* relies on observing the actual successor states of action executions, it can have computational advantages even over several search episodes compared to a complete minimax search. This is the case if nature is not as malicious as a minimax search assumes and some successor states do not occur in practice. This also means that Min-Max LRTA* might be able to solve planning tasks in domains that are not safely explorable, although this is not guaranteed. Section 6.4 presents repeated runs of Min-Max LRTA* until convergence in the context of robot-navigation tasks, demonstrating how Min-Max LRTA* can improve its performance in case its heuristic knowledge does not guide planning sufficiently well at first.

6 Application Example: Robot-Navigation Tasks

In this section, we present a case study that illustrates the application of Min-Max LRTA* to simulated robot-navigation tasks [26]. We study robot-navigation tasks with initial pose uncertainty, an example of which we showed in Figure 1. A robot knows the maze, but is uncertain about its start pose, where a *pose* is a location (square) and orientation (north, east, south, west). The sensors on-board the robot tell it in every pose whether there are walls immediately adjacent to it in the four directions (front, left, behind, right). The actions are to move forward one square (unless there is a wall directly in front of the robot), turn left ninety degrees, or turn right ninety degrees. We assume that there is no uncertainty in actuation and sensing. This assumption has been shown to be sufficiently close to reality to enable one to use search methods developed under this assumption on actual robots. Nourbakhsh [39] pioneered their application in robot programming classes where Nomad 150 mobile robots had to navigate mazes that were built with three-foot high and forty inch long cardboard walls. The success rate of turning left or right was reported as 100.00 percent in these environments, the success rate of moving forward was at least 99.57 percent, and the success rate of sensing walls in all four directions simultaneously was at least 99.38 percent [40]. We study two different robot-navigation tasks. Localization tasks require the robot to achieve certainty about its pose. Goal-directed navigation tasks, on the other hand, require the robot to navigate to any of the given goal poses and stop. Since there might be many poses that produce the same sensor reports as the goal poses, solving goal-directed navigation tasks includes localizing the robot sufficiently so that it knows that it is at a goal pose when it stops. Robot navigation tasks with initial pose uncertainty are good tasks for interleaving planning and plan executions because interleaving planning and plan executions allows the robot to gather information early, which reduces its pose uncertainty and thus the number of situations that its plans have to cover. This makes subsequent planning more efficient.

We require that the mazes be strongly connected (every pose can be reached from every other pose) and not completely symmetrical (localization is possible). This modest assumption makes all robot navigation tasks

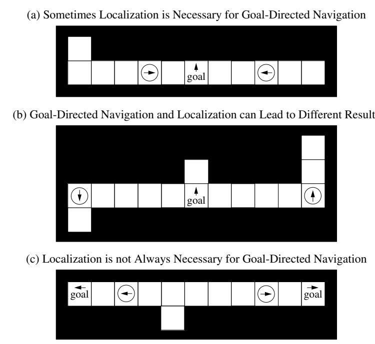


Figure 8: Goal-Directed Navigation and Localization

solvable, since the robot can always first localize itself and then, for goal-directed navigation tasks, move to a goal pose. It can indeed be optimal for the robot to solve goal-directed navigation tasks this way. Figure 8(a) shows an example. Since the observations at the goal pose do not uniquely identify the goal pose, it is best for the robot to first localize itself by turning around and moving forward twice. At this point, the robot has localized itself and can navigate to the goal pose. However, it is not always optimal for the robot to solve goal-directed navigation tasks this way. Figure 8(b) shows an example. To solve the goal-directed navigation task, it is best for the robot to turn left and move forward until it sees a corridor opening on one of its sides. At this point, the robot has localized itself and can navigate to the goal pose. On the other hand, to solve the corresponding localization task, it is best for the robot to move forward once. At this point, the robot has localized itself. Finally, some goal-directed navigation tasks can be solved without localizing the robot at all. Figure 8(c) shows an example. To solve the goal-directed navigation task, it is best for the robot to move forward twice. At this point, the robot knows that it is in a goal pose but cannot be certain which of the two goal poses it is in.

At every point in time, the robot has a certain pose but, due to the partial observability of the poses, might not know what its current pose is. However, the robot can always maintain a belief about its current pose. We assume that the robot cannot associate probabilities or other likelihood estimates with the poses. Then, all it can do is maintain a set of possible poses, namely the poses that it could possibly be in. For example, if the robot has no knowledge of its start pose for the robot-navigation task from Figure 1, but observes walls around it except in its front, then the start belief of the robot contains the seven possible start poses shown in the figure. We can use Min-Max LRTA* in the state space of beliefs to solve the robot-navigation tasks.

6.1 Formalizing the Robot-Navigation Tasks

In this section, we formally describe the robot-navigation tasks and the state space that Min-Max LRTA* searches to solve them. We use the following notation: P is the finite set of possible robot poses (pairs of location and orientation). $A(p)$ is the set of possible actions that the robot can execute in pose $p \in P$: left, right, and possibly forward. $succ(p, a)$ is the pose that results from the execution of action $a \in A(p)$ in pose $p \in P$. $o(p)$ is the observation that the robot makes in pose $p \in P$: whether or not there are walls immediately adjacent to it in the four directions (front, left, behind, right). The robot starts in pose $p_{start} \in P$ and then repeatedly makes an observation and executes an action until it decides to stop. It knows the maze, but is uncertain about its start pose. It could be in any pose in $P_{start} \subseteq P$. We require only that $o(p) = o(p')$ for all $p, p' \in P_{start}$, which automatically holds after the first observation, and $p_{start} \in P_{start}$, which automatically holds for $P_{start} = \{p : p \in P \wedge o(p) = o(p_{start})\}$. B is the set of beliefs and b_{start} the start belief. $A(b)$ is the set of actions that can be executed when the belief is b. $O(b, a)$ is the set of possible observations that can be made after the execution of action a when the belief was b. $succ(b, a, o)$ is the successor belief that results if observation o is made after the execution of action a when the belief was b. Then, for all $b \in B$, $a \in A(b)$, and $o \in O(b, a)$,

$$
\begin{aligned}
B &= \{b : b \subseteq P \,\wedge\, o(p) = o(p') \text{ for all } p, p' \in b\} \\
b_{start} &= P_{start} \\
A(b) &= A(p) \text{ for any } p \in b \\
O(b,a) &= \{o(succ(p,a)) : p \in b\} \\
succ(b,a,o) &= \{succ(p,a) : p \in b \,\wedge\, o(succ(p,a)) = o\}
\end{aligned}
$$

To understand the definition of $A(b)$, notice that $A(p) = A(p')$ for all $p, p' \in b$ after the preceding observation since the observation determines the actions that can be executed.

For goal-directed navigation tasks, the robot has to navigate to any pose in $\emptyset \neq P_{goal} \subseteq P$ and stop. In this case, we define the set of goal beliefs as $B_{goal} = \{b : b \subseteq P_{goal} \,\wedge\, o(p) = o(p') \text{ for all } p, p' \in b\}$. To understand this definition, notice that the robot knows that it is in a goal pose if its belief is $b \subseteq P_{goal}$. If the belief contains more than one pose, however, the robot does not know which goal pose it is in. An example was discussed in the context of Figure 8(c). If it is important that the robot knows which goal pose it is in, we use $B_{goal} = \{b : b \subseteq P_{goal} \,\wedge |b| = 1\}$. For localization tasks, we use $B_{goal} = \{b : b \subseteq P \,\wedge |b| = 1\}$.

The robot-navigation domain is deterministic and small (*pose space*) when the robot knows its pose. However, the beliefs of the robot depend on its observations, which the robot cannot predict with certainty if it is uncertain about its pose. For example, it cannot predict whether there will be a wall in front of it after it moves forward for the goal-directed navigation task shown in Figure 1. This explains why the robot-navigation tasks are nondeterministic planning tasks despite the lack of actuator and sensor noise. We therefore formulate them as planning tasks in a nondeterministic domain whose states are the beliefs of the robot (*belief space*). Beliefs are sets of poses. Thus, the number of beliefs is exponential in the number of poses, and the belief space is not only nondeterministic but can also be large. The pose space and the belief space differ in the observability of their states. The robot will usually be uncertain about its current pose but can always determine its current belief for sure. Thus, we can characterize the belief space as follows

$$
\begin{aligned}
S &= B \\
s_{start} &= b_{start} \\
G &= B_{goal} \\
A(s) &= A(b) \text{ for } s = b \\
succ(s,a) &= \{succ(b,a,o) : o \in O(b,a)\} \text{ for } s = b.
\end{aligned}
$$

The actual belief that results from the execution of action a in belief $s = b$ is $succ(b,a,o)$ if observation o is made after the action execution.

Planning in the belief space satisfies the description of our planning tasks from Section 2, and the belief space is safely explorable according to our assumptions. Thus, we can use Min-Max LRTA* to search it and solve the robot-navigation tasks, that is, move the robot so that it achieves certainty about its pose for localization tasks or stops in a goal pose for goal-directed navigation tasks. In general, traversing any path from the start state in the belief space to a goal state solves the robot-navigation task. The path does not have to be optimal or repeatable. Assume, for example, that the robot has successfully solved some robot-navigation task and now has to solve the same kind of robot-navigation task in the same maze with the same start and goal beliefs. Even if the robot attempts to execute the same plan again, it can make different observations since, unknown to the robot, its start pose might have changed. This is possible since many start poses can be consistent with the same start belief. The change can result in different beliefs during plan execution and thus different trajectories through the domain. Since there was no reason to plan for the new beliefs, the previous plan might not cover them.

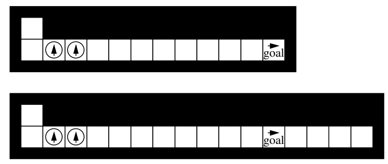


Figure 9: Goal-Directed Navigation Task 1

6.2 Features of Min-Max LRTA* for Robot-Navigation Tasks

Earlier, we discussed the following features of Min-Max LRTA*: that it uses heuristic knowledge to guide planning, allows for fine-grained control over how much planning to do between plan executions, and improves its plan-execution time as it solves similar planning tasks until its plan-execution time is at least worst-case optimal. In this section, we discuss these features in the context of the robot-navigation tasks.

6.2.1 Heuristic Knowledge

Min-Max LRTA* uses heuristic knowledge to guide planning. This knowledge is provided in the form of admissible initial u-values. For goal-directed navigation tasks, one can use the *goal-distance heuristic* to initialize the u-values, that is, $u(s) = \max_{p \in s} gd(\{p\})$. The calculation of $gd(\{p\})$ involves no pose uncertainty and can be done efficiently without interleaving planning and plan executions, by using traditional search methods in the pose space. This is possible because the pose space is deterministic and small. The u-values are admissible because the robot needs at least $\max_{p \in s} gd(\{p\})$ action executions in the worst case to solve the goal-directed navigation task from pose $p' =$ one-of $\arg\max_{p \in s} gd(\{p\})$, even if it knows that it starts in that pose. The u-values are often only partially informed because they do not take into account that the robot might not know its pose and then might have to execute additional localization actions to overcome its pose uncertainty. For localization tasks, on the other hand, it is difficult to obtain better informed initial u-values than zero-initialized ones.

6.2.2 Fine-Grained Control

Min-Max LRTA* allows for fine-grained control over how much planning to do between plan executions. We have argued earlier that look-aheads of one or two action executions can be optimal for sufficiently fast acting agents, that is, agents that can execute plans with a similar speed as they can generate them. Robots, however, cannot execute actions that fast and thus larger local search spaces can be expected to outperform small local search spaces.

6.2.3 Improvement of Plan-Execution Time

Min-Max LRTA* can improve its plan-execution time by transferring domain knowledge between goal-directed navigation tasks with the same goal poses in the same maze and between localization tasks in the same maze. The actual start poses or the beliefs of the robot about its start poses do not need to be identical. Figure 9 demonstrates why this is important for the robot-navigation tasks. Whether the robot should localize right away by moving left and then forward one square depends on features of the mazes far away from the currently possible poses. In particular, the robot should localize right away in the bottom maze but should turn right and move forward repeatedly in the top maze since it will automatically localize at the goal location in the top maze. However, search methods with limited look-ahead cannot treat these situations differently. Assume, for example, that the robot repeatedly solves the same robot-navigation task with the same start pose but does not know that its start pose remains the same. Then, the behavior of nature does not change since it is completely determined by the actual start pose of the robot, that remains the same for all tasks. Thus, nature cannot exhibit the behavior described in Section 5.3 and fool Min-Max LRTA* for an arbitrarily long time. Assume further

1. $S_{lss} := \{s\}$.
2. Update $u(s)$ for all $s \in S_{lss}$ (Figure 5).
3. $s' := s$.
4. $a :=$ one-of arg $\min_{a' \in A(s')} \max_{s'' \in succ(s',a')} u(s'')$.
5. If $|succ(s', a)| > 1$, then return.
6. $s' := s''$, where $s'' \in succ(s', a)$ is unique.
7. If $s' \in S_{lss}$, then go to 4.
8. If $s' \in G$, then return.
9. $S_{lss} := S_{lss} \cup \{s'\}$ and go to 2.

Figure 10: Generating Local Search Spaces

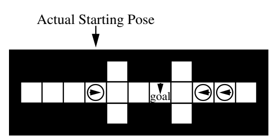


Figure 11: Goal-Directed Navigation Task 2

that the way Min-Max LRTA* selects its local search spaces does not change and that Line 5 in Figure 4 breaks ties systematically according to a predetermined ordering on $A(s)$ for all states s. Then, once the u-values do not change during one robot-navigation task, they cannot change during any future robot-navigation task because the behavior of Min-Max LRTA* is now completely determined by the u-values and the behavior of nature. Since the u-values are monotonically nondecreasing and bounded from above by the minimax goal distances, which are finite in safely explorable domains, the u-values and thus also the executed action sequence converge after a bounded number of action executions (or planning tasks).

6.3 Alternative Approaches for Robot-Navigation Tasks

Min-Max LRTA* is a domain-independent planning method that does not only apply to robot-navigation tasks with initial pose uncertainty, but to planning tasks in nondeterministic domains in general, which has been investigated in [6]. Planning methods in nondeterministic domains that interleave planning and plan executions have been studied before. This includes assumptive planning, deliberation scheduling (including anytime algorithms), on-line algorithms and competitive analysis, reinforcement learning, robot exploration techniques, and sensor-based planning. The proceedings of the AAAI-97 Workshop on On-Line Search [24] give a good overview of these and other techniques. Real-time heuristic search methods are also related to forward-chaining planners and local search techniques from combinatorial search and constraint satisfaction (including simulated annealing, tabu search, hill climbing, some SAT solution methods, and some scheduling methods) [50].

In this section, we compare Min-Max LRTA* to two more specialized planning methods that can also be used to solve robot-navigation tasks (but not nondeterministic planning tasks in general). An important advantage of Min-Max LRTA* over these planning methods is that it can improve its plan-execution time as it solves similar planning tasks.

6.3.1 Information-Gain Method

The *Information-Gain Method* (IG method) [13] first demonstrated the advantage of interleaving planning and plan executions in the context of robot-navigation tasks.[3] It uses breadth-first search (iterative deepening) around the current state in conjunction with pruning rules to find subplans that achieve a gain in information, in the following sense: after the execution of the subplan it is guaranteed that the robot has either solved the

[3] [13] refers to the IG method as the Delayed Planning Architecture (DPA) with the viable plan heuristic. It also contains some improvements on the version of the IG method discussed here, that do not change its character.

robot-navigation task or at least reduced the number of poses it can be in. This way, the IG method guarantees progress towards a solution.

There are similarities between the IG method and Min-Max LRTA*: Both planning methods interleave searches and plan executions. Min-Max LRTA* generalizes the IG method since zero-initialized Min-Max LRTA* that generates the local search spaces for robot-navigation tasks with the method from Figure 10 exhibits a similar behavior as the IG method before the u-values of Min-Max LRTA* become more informed: it also performs a breadth-first search around its current state until it finds a subplan whose execution results in a gain in information. The method does this by starting with the local search space that contains only the current state. It performs a minimax search in the local search space and then simulates the action executions of Min-Max LRTA* starting from its current state. If the simulated action executions reach a goal state or lead to a gain in information, then the method returns. However, if the simulated action executions leave the local search space without reaching a goal state or leading to a gain in information, the method halts the simulation, adds the state outside of the local search space to the local search space, and repeats the procedure. Notice that, when the method returns, it has already updated the u-values of all states of the local search space. Thus, Min-Max LRTA* does not need to improve the u-values of these states again and can skip the minimax search. Its action-selection step (Line 5) and the simulation have to break ties identically. Then, Min-Max LRTA* with Line 8 in Figure 4 executes actions until it either reaches a goal state or gains information. Notice that this method basically performs a search in the deterministic part of the state space around the current state. This property could be used to simplify the method further and make it (almost) identical to traditional search methods. As an example, consider again the goal-directed navigation tasks shown in Figure 9. In the bottom maze, it is best for the robot to gain information as quickly as possible. Thus, both the IG method and zero-initialized Min-Max LRTA* that generates the local search spaces with the method from Figure 10 behave in a worst-case optimal way. On the other hand, in the top maze, it is best for the robot not to gain information as quickly as possible but to move towards the goal pose and gain information in the process. Thus, both the IG method and zero-initialized Min-Max LRTA* that generates the local search spaces with the method from Figure 10 behave in a suboptimal way. Section 6.4 presents a complete example run of Min-Max LRTA* that generates the local search spaces for robot-navigation tasks with the method from Figure 10.

There are also differences between the IG method and Min-Max LRTA*: Min-Max LRTA* can take advantage of heuristic knowledge to guide planning, although the IG method could be augmented to do so as well. This is often an advantage when solving goal-directed navigation tasks because it allows the robot to move towards the goal while localizing sufficiently. A more important difference between the IG method and Min-Max LRTA* is that the IG method does not need to maintain information between plan executions, whereas Min-Max LRTA* has to maintain information in the form of u-values. To save memory, Min-Max LRTA* can generate the initial u-values on demand and never store u-values that are identical to their initial values. Even then its memory requirements are bounded only by the number of states, but Section 6.5 shows that they appear to be small in practice. The u-values allow Min-Max LRTA* both to adapt its look-ahead better to agents that can execute plans with a similar speed as they can generate them and to improve its plan-execution time as it solves similar planning tasks. The u-values allow Min-Max LRTA* to adapt its look-ahead better to agents that can execute plans with a similar speed as they can generate them because they allow Min-Max LRTA* to use smaller local search spaces than the IG method. In particular, Min-Max LRTA* can use local search spaces that do not guarantee a gain in information because the increase of the potential $\sum_{s \in S \setminus \{s^t\}} u^t(s)$ from Section 4.3 serves as a (virtual) gain in information. Since robots move slowly compared to their speed of computation, a more important advantage of Min-Max LRTA* over the IG method is that it can improve its plan-execution time as it solves similar planning tasks. This property is important because no planning method that interleaves planning and plan executions can guarantee a good plan-execution time on the first run. For instance, consider the goal-directed navigation task from Figure 11 and assume that Min-Max LRTA* generates the local search spaces with the method from Figure 10. Then, both the IG method and zero-initialized Min-Max LRTA* move forward, because this is the fastest way to eliminate a possible pose, that is, to gain information. Even Min-Max LRTA* with the goal-distance heuristic moves forward, since it follows the gradient of the u-values. However, moving forward is suboptimal. It is best for the robot to first localize itself by turning around and moving to a corridor end. If the goal-directed navigation task is repeated a sufficient number of times, Min-Max LRTA* eventually learns this behavior.

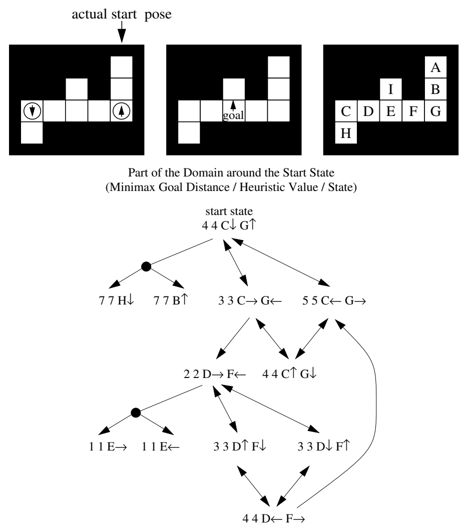


Figure 12: Goal-Directed Navigation Task 3

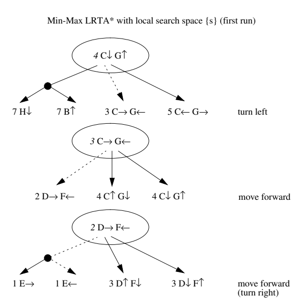


Figure 13: Behavior of Min-Max LRTA*

6.3.2 Homing Sequences

Localization tasks are related to finding homing sequences or adaptive homing sequences for deterministic finite state automata whose states are colored. A *homing sequence* is a linear plan (action sequence) with the property that the observations made during its execution uniquely determine the resulting state [28]. An adaptive homing sequence is a conditional plan with the same property [49]. For every reduced deterministic finite state automaton, there exists a homing sequence that contains at most $(n-1)^2$ actions. (A reduced finite state automaton is one where, for every pair of different states, there exists some action sequence that distinguishes them.) Finding a shortest homing sequence is NP-complete in general but a suboptimal homing sequence of at most $(n-1)^2$ actions can be found in polynomial time [49]. Robot localization tasks can be solved with homing sequences since the pose space is deterministic and thus can be modeled as a deterministic finite state automaton. Homing sequences can be used to achieve the same behavior as the IG method.

6.4 Examples of Min-Max LRTA* for Robot-Navigation Tasks

In this section, we provide examples of how Min-Max LRTA* with Line 8 and the goal-distance heuristic solves goal-directed navigation tasks. We use a version of Min-Max LRTA* that breaks ties between actions systematically according to a pre-determined ordering on $A(s)$ for all states s. It breaks ties in the following order (from highest to lowest priority): move forward, turn left, and turn right. We stop the search once the robot has localized itself since the planning task then becomes deterministic and can be solved greedily by using steepest descent on the goal-distance heuristic. (In the figures, we put these actions in parentheses.)

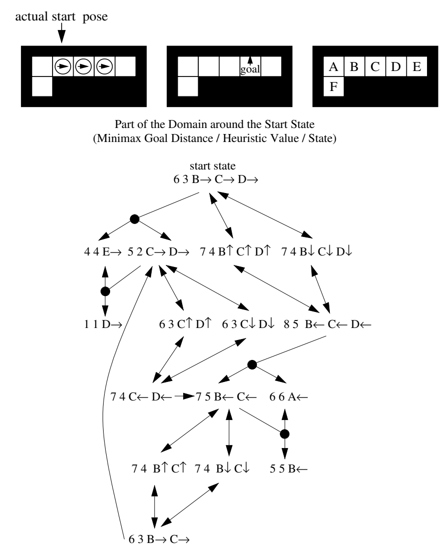


Figure 14: Goal-Directed Navigation Task 4

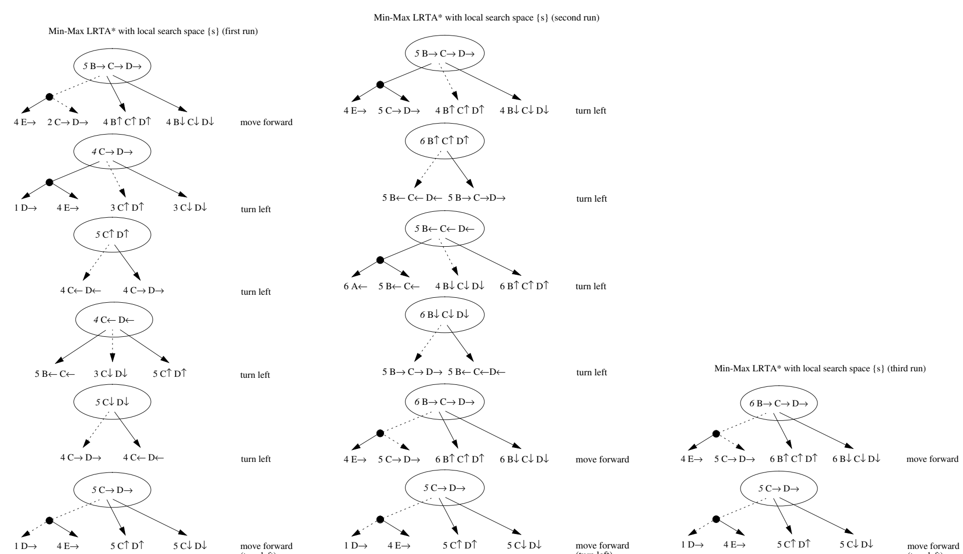


Figure 15: Behavior of Min-Max LRTA*

Heuristic knowledge often guides planning well right away. To demonstrate the behavior of Min-Max LRTA* in this case, we use the goal-directed navigation task from Figure 12, that is similar to the goal-directed navigation task from Figure 8(b). The goal-distance heuristic predicts the minimax goal distances perfectly in the relevant part of the state space. Figure 13 shows how Min-Max LRTA* with local search spaces $S_{lss} = \{s\}$ solves the goal-directed navigation task. The robot localizes towards the goal and behaves right away in a worst-case optimal way. The robot gains information only after three action executions. A robot that always executes a shortest action sequence that results in a gain in information starts by moving forward, which results in a gain in information right away but is suboptimal. If Min-Max LRTA* solves the same goal-directed navigation task again with the same start pose, it continues to execute the same actions since it does not change any u-value during the first run.

Sometimes the heuristic knowledge does not guide planning well at first. To demonstrate the behavior of Min-Max LRTA* in this case, we use the goal-directed navigation task from Figure 14. The goal-distance heuristic often underestimates the minimax goal distances a lot in the relevant part of the state space. Figure 15 (left)

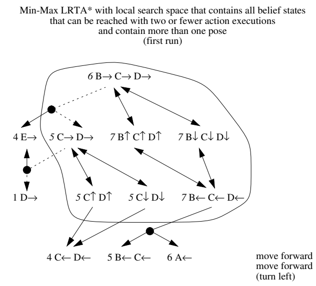


Figure 16: Behavior of Min-Max LRTA*

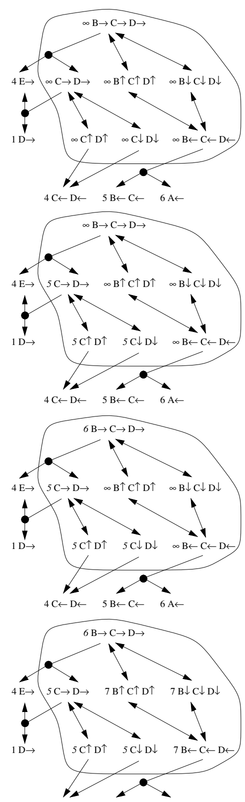


Figure 17: Minimax Search

shows how Min-Max LRTA* with local search spaces $S_{lss} = \{s\}$ solves the goal-directed navigation task. The robot spins in place once after it has moved forward but reaches the goal pose eventually. This demonstrates how updating the u-values ensures that Min-Max LRTA* cannot get stuck in cycles. For example, Min-Max LRTA* initially moves to belief $\{C \uparrow, D \uparrow\}$ when it is in belief $\{C \rightarrow, D \rightarrow\}$ because the u-value of belief $\{C \uparrow, D \uparrow\}$ is three. That u-value changes during the third minimax search to five, preventing Min-Max LRTA* from moving to belief $\{C \uparrow, D \uparrow\}$ when it is in belief $\{C \rightarrow, D \rightarrow\}$ again. Figure 15 (center) shows how Min-Max LRTA* solves the goal-directed navigation task a second time with the same start pose. (The robot does not know that the start pose remains the same. Otherwise, it could use the decreased uncertainty about its pose after solving the goal-directed navigation task to narrow down its start pose and improve its plan-execution time this way.) The robot still spins in place once, this time before it moves forward. Figure 15 (right) shows how Min-Max LRTA* solves the goal-directed navigation task a third time with the same start pose. The robot now behaves in a worst-case optimal way. If Min-Max LRTA* solves the same goal-directed navigation task again with the same start pose, it continues to execute the same actions since it does not change any u-value during the third run, demonstrating how updating the u-values guarantees that the robot eventually behaves in

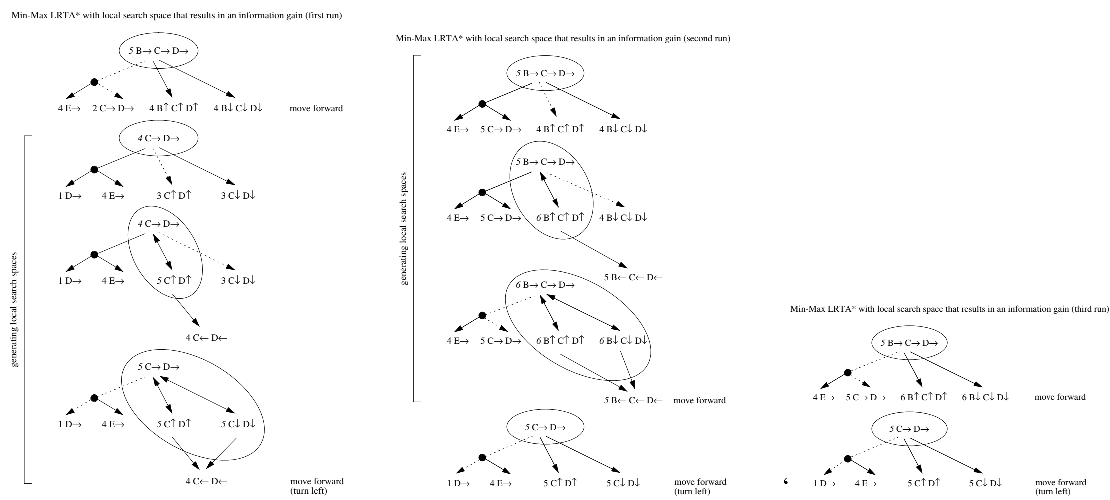


Figure 18: Behavior of Min-Max LRTA*

a way that is at least worst-case optimal even if the heuristic knowledge does not guide planning well at first. Figure 16 shows how Min-Max LRTA* with local search spaces that contain all beliefs that can be reached with two or fewer action executions and contain more than one pose solves the goal-directed navigation task. The robot behaves right away in a worst-case optimal way, demonstrating that larger local search spaces can compensate for deficiencies of the heuristic knowledge. Figure 17 shows in detail how the minimax-search method from Figure 5 determines the u-values of all states in the (initial) local search space. The minimax search method assigns values to states sequentially. In the figure, assignments of the same value to states have been grouped together. If Min-Max LRTA* solves the same goal-directed navigation task again with the same start pose, it continues to execute the same actions since it does not change any u-value during the first run. Finally, Figure 18 (left) shows how Min-Max LRTA* that generates the local search spaces with the method from Figure 10 solves the goal-directed navigation task. Again, the robot behaves right away in a worst-case optimal way, confirming that increasing the local search spaces dynamically can help to soften the problems with misleading heuristic knowledge during the first run [18]. Figure 18 (center) shows how Min-Max solves the goal-directed navigation task a second time with the same start pose, and Figure 18 (right) shows how Min-Max LRTA* solves it a third time. In both cases, the robot continues to behave in a worst-case optimal way although the local search spaces change. If Min-Max LRTA* solves the same goal-directed navigation task a fourth time with the same start pose, it continues to execute the same actions since it does not change any u-value during the third run. The local search spaces are large initially but eventually become small. Consequently, the planning time decreases.

The heuristic knowledge can be even more misleading although this happened only rarely during the experiments. We apply Min-Max LRTA* with local search spaces $S_{lss} = \{s\}$ to the goal-directed navigation task from Figure 19. The goal is a location, which includes all four poses. Since the value surface of the u-values has a depression around the start state, Min-Max LRTA* has to move backward and forward to increase the u-values and leave this part of the state space. During the first run, Min-Max LRTA* can traverse the following state sequence

$$\begin{array}{lllllll}
\{E\rightarrow, G\rightarrow\}, & \{E\downarrow, G\downarrow\}, & \{E\leftarrow, G\leftarrow\}, & \{E\uparrow, G\uparrow\}, & \{E\rightarrow, G\rightarrow\}, & \{F\rightarrow, H\rightarrow\}, & \{F\downarrow, H\downarrow\}, \\
\{F\leftarrow, H\leftarrow\}, & \{E\leftarrow, G\leftarrow\}, & \{D\leftarrow, F\leftarrow\}, & \{D\uparrow, F\uparrow\}, & \{D\rightarrow, F\rightarrow\}, & \{D\downarrow, F\downarrow\}, & \{D\leftarrow, F\leftarrow\}, \\
\{D\uparrow, F\uparrow\}, & \{D\rightarrow, F\rightarrow\}, & \{E\rightarrow, G\rightarrow\}, & \{F\rightarrow, H\rightarrow\}, & \{F\downarrow, H\downarrow\}, & \{F\leftarrow, H\leftarrow\}, & \{F\uparrow, H\uparrow\}, \\
\{F\rightarrow, H\rightarrow\}, & \{G\rightarrow, I\rightarrow\}, & \{G\downarrow, I\downarrow\}, & \{G\leftarrow, I\leftarrow\}, & \{F\leftarrow, H\leftarrow\}, & \{E\leftarrow, G\leftarrow\}, & \{D\leftarrow, F\leftarrow\}, \\
\{C\leftarrow, E\leftarrow\}, & \{C\uparrow, E\uparrow\}, & \{C\rightarrow, E\rightarrow\}, & \{C\downarrow, E\downarrow\}, & \{C\leftarrow, E\leftarrow\}, & \{C\uparrow, E\uparrow\}, & \{C\rightarrow, E\rightarrow\}, \\
\{D\rightarrow, F\rightarrow\}, & \{E\rightarrow, G\rightarrow\}, & \{E\downarrow, G\downarrow\}, & \{E\leftarrow, G\leftarrow\}, & \{E\uparrow, G\uparrow\}, & \{E\rightarrow, G\rightarrow\}, & \{F\rightarrow, H\rightarrow\}, \\
\{G\rightarrow, I\rightarrow\}, & \{G\downarrow, I\downarrow\}, & \{G\leftarrow, I\leftarrow\}, & \{G\uparrow, I\uparrow\}, & \{G\rightarrow, I\rightarrow\}, & \{H\rightarrow, J\rightarrow\}, & \{H\downarrow, J\downarrow\}, \\
\{H\leftarrow, J\leftarrow\}, & \{G\leftarrow, I\leftarrow\}, & \{F\leftarrow, H\leftarrow\}, & \{E\leftarrow, G\leftarrow\}, & \{D\leftarrow, F\leftarrow\}, & \{C\leftarrow, E\leftarrow\}, & \{B\leftarrow, D\leftarrow\}, \\
\{B\uparrow, D\uparrow\}, & \{B\rightarrow, D\rightarrow\}, & \{B\downarrow, D\downarrow\}, & \{B\leftarrow, D\leftarrow\}, & \{B\uparrow, D\uparrow\}, & \{B\rightarrow, D\rightarrow\}, & \{C\rightarrow, E\rightarrow\}, \\
\{D\rightarrow, F\rightarrow\}, & \{D\downarrow, F\downarrow\}, & \{D\leftarrow, F\leftarrow\}, & \{D\uparrow, F\uparrow\}, & \{D\rightarrow, F\rightarrow\}, & \{E\rightarrow, G\rightarrow\}, & \{F\rightarrow, H\rightarrow\}, \\
\{F\downarrow, H\downarrow\}, & \{F\leftarrow, H\leftarrow\}, & \{F\uparrow, H\uparrow\}, & \{F\rightarrow, H\rightarrow\}, & \{G\rightarrow, I\rightarrow\}, & \{H\rightarrow, J\rightarrow\}, & \{H\downarrow, J\downarrow\}, \\
\{H\leftarrow, J\leftarrow\}, & \{H\uparrow, J\uparrow\}, & \{H\rightarrow, J\rightarrow\}. & & & &
\end{array}$$

After the next action execution, the robot has localized itself and can navigate to the goal location with seven action executions. In this case, the robot executes 87 actions before it reaches a goal state. If Min-Max LRTA* solves the same goal-directed navigation task again with the same start pose, the robot eventually learns to

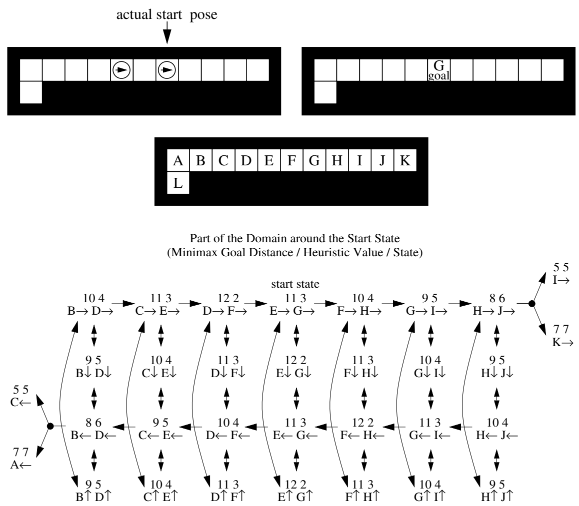


Figure 19: Goal-Directed Navigation Task 5

after ...	measuring ...	using ...	Min-Max LRTA* with local search space $S_{lss} = \{s\}$		Min-Max LRTA* with larger local search space (using the method from Figure 10)	
			goal-directed navigation goal-distance heuristic	localization zero heuristic	goal-directed navigation goal-distance heuristic	localization zero heuristic
the first run	plan-execution time	action executions	113.32	13.33	50.48	12.24
	planning time	state expansions	113.32	13.33	73.46	26.62
	memory usage	u-values remembered	31.88	13.32	30.28	26.62
convergence	plan-execution time	action executions	49.15	8.82	49.13	8.81
	planning time	state expansions	49.15	8.82	49.13	8.81
	memory usage	u-values remembered	446.13	1,782.26	85.80	506.63
number of runs until convergence			16.49	102.90	3.14	21.55

Figure 20: Experimental Results

move forward repeatedly until it has localized itself and then navigate to the goal location. The robot then executes only eleven actions before it reaches a goal state and behaves in a worst-case optimal way. Min-Max LRTA* converges to this behavior in less than ten runs.

6.5 Experimental Results for Robot-Navigation Tasks

We have argued that interleaving planning and plan executions often decreases the planning time. On the other hand, it also increases the number of action executions because the agent executes actions before it knows their complete consequences and thus cannot be sure that their execution is really as advantageous as anticipated. This increases the plan-execution time and could potentially also increase the planning time because the agent has to plan more often. The strength of this effect depends on the speed of the agent, how well the heuristic knowledge guides planning, and similar factors. In this section, we therefore perform experiments to study the behavior of Min-Max LRTA* for the robot navigation tasks.

Previous work reported experimental [39] and theoretical [56] evidence that performing a complete minimax search to solve robot-navigation tasks in a worst-case optimal way can be completely infeasible. We show in this section that Min-Max LRTA* solves the robot-navigation tasks fast, converges quickly, and requires only a small amount of memory. We do this experimentally since the actual plan-execution time of Min-Max LRTA* and its memory requirements can be much smaller than the upper bound of Theorem 4 suggests. We use a simulation of the robot-navigation tasks whose interface matches the interface of an actual robot that operates in mazes [42]. Thus, Min-Max LRTA* could be run on that robot.

We use Min-Max LRTA* as described in the previous section, with local search spaces of two different sizes each, namely local search spaces $S_{lss} = \{s\}$ and the larger local search spaces from Figure 10. To save memory, Min-Max LRTA* generates the initial u-values only on demand and never stores u-values that are

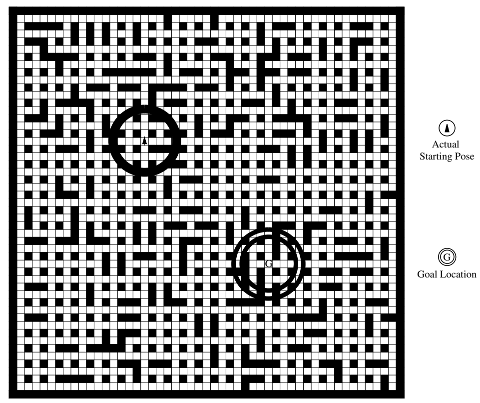


Figure 21: Sample Maze

identical to their initial values.

As test domains, we use 500 randomly generated square mazes. The same 500 mazes are used for all experiments. All mazes have size 49×49 and the same obstacle density, the same start pose of the robot, and (for goal-directed navigation tasks) the same goal location, which includes all four poses. Figure 21 shows an example. We provide Min-Max LRTA* with no further knowledge of its actual start pose and let it solve the same robot-navigation task repeatedly with the same start pose until its behavior converges. We use the same start pose repeatedly because this way Min-Max LRTA* converges after a finite number of action executions and convergence can easily be detected when the u-values do not change any longer during a robot-navigation task, as described in Section 5.3. Of course, the robot does not know that the start pose remains the same because otherwise it could use the decreased uncertainty about its pose after solving the robot-navigation task to narrow down its start pose and improve its plan-execution time this way. Since the robot has no additional knowledge, it must assume that it could be in any pose that is consistent with its first observation after the re-start.

Figure 20 shows that Min-Max LRTA* indeed produces good plans for robot-navigation tasks with a small number of node expansions, while using only a small amount of memory.

Since the plan-execution time of Min-Max LRTA* after convergence is no worse than the minimax goal distance of the start state, we know that its initial plan-execution time is at most 231, 151, 103, and 139 percent (respectively) of the worst-case optimal plan-execution time for the given start belief if nature can choose the start pose freely. For example, the third column in Figure 20 shows that Min-Max LRTA* that generates the local search spaces for robot-navigation tasks with the method from Figure 10 and the goal-distance heuristic solves goal-directed navigation tasks on average with 50.48 action executions on the first run, and with 49.13 action executions after convergence. Thus, the minimax goal distance of the start state is at least 49.13 action executions, and the initial plan-execution time of Min-Max LRTA* is at most $50.48/49.13 \times 100 \approx 103$ percent of the worst-case optimal plan-execution time. Thus, in this case, the plan-execution time is close to worst-case optimal right away.

Similarly, the first column in Figure 20 shows that Min-Max LRTA* with local search spaces $S_{lss} = \{s\}$ and the goal-distance heuristic solves goal-directed navigation tasks on average with 113.32 action executions on the first run, and with 49.15 action executions after convergence. In this case, Min-Max LRTA* converges quickly. It more than halves its plan-execution time in less than twenty runs. This demonstrates that this aspect of Min-Max LRTA* is important if the heuristic knowledge does not guide planning sufficiently well at first. The total run-time until convergence is 6.85 seconds for our straight-forward unoptimized Common Lisp implementation of Min-Max LRTA* on a Pentium II 300 MHz computer equipped with 192 MByte RAM, for an amortized run time of 0.42 seconds per run. The convergence times can be expected to increase substantially as the relevant part of the state space gets larger, for example, if the start pose of the robot is not always the same.

We have used Min-Max LRTA* with mazes that were as large as 249×249. However, when we applied Min-Max LRTA* with local search spaces $S_{lss} = \{s\}$ and the goal-distance heuristic to goal-directed navigation tasks, a large number of garbage collections increased the amortized run time from 0.42 seconds per run to 54.36 seconds per run. This is partly due to an increase of the average number of poses per belief state. For example, the average number of poses that are consistent with the initial observation of seeing openings in all four directions increases from about 800 to more than 20,000 poses.

Finally, we compare Min-Max LRTA* to planning methods that always execute the shortest action sequences that result in a gain in information.

First, we demonstrate the advantage of using heuristic knowledge to guide planning. We compare the number of action executions on the first run of two planning methods for goal-directed navigation tasks, namely Min-Max LRTA* that generates the local search spaces for robot-navigation tasks with the method from Figure 10 and planning methods that always execute the shortest action sequences that result in a gain in information. Both planning methods need about the same number of action executions if Min-Max LRTA* uses the zero heuristic. However, Min-Max LRTA* has an advantage of about three action executions if it uses the goal-distance heuristic due to the guidance of the heuristic knowledge.

Second, we demonstrate the advantage of improving the plan-execution time with experience. We compare the number of action executions of two planning methods for localization tasks, namely Min-Max LRTA* that generates the local search spaces for robot-navigation tasks with the method from Figure 10 and the zero heuristic and planning methods that always execute the shortest action sequences that result in a gain in information. Both methods need on average around twelve action executions on the first run but Min-Max LRTA* is able to reduce this number to around nine action executions and then has an advantage of about three actions executions due to its capability to learn from experience. This number will be larger in mazes in which localization is harder than in random mazes, and the results in [56] give some evidence on how to construct such mazes.

6.6 Extensions of Min-Max LRTA* for Robot-Navigation Tasks and Future Work

In this section, we discuss disadvantages of Min-Max LRTA*. Addressing them is future work, but in the following we briefly outline possible extensions of Min-Max LRTA* that remedy some of the problems.

The memory consumption of Min-Max LRTA* can be large in principle since it can assign a u-value to every state. We demonstrated experimentally that the memory consumption of Min-Max LRTA* appears to be small in practice but it is currently unknown exactly where its breaking-point is. For example, for robot-navigation tasks, each state is a set of poses. Thus, the amount of memory consumed by the representation of one belief already grows linearly in the size of the maze and the number of beliefs even grows exponentially. This does not only increase the memory requirements of Min-Max LRTA* but also its planning time. It also increases the likelihood that Min-Max LRTA* cannot improve its plan-execution time with experience because it never visits the neighborhoods of previously visited beliefs again. In this case, Min-Max LRTA* needs more powerful generalization capabilities that allow it to infer the u-values of states from the u-values of similar states. Such a generalization capability would not only decrease the time to convergence but already the plan-execution time of the first run. For robot-navigation tasks it is possible to combine the local searches with updates over a greater distance to achieve a very weak generalization capability, with only a small amount of additional effort. For example, we know that $gd(s) \geq gd(s')$ for any two states $s, s' \in S$ with $s \supseteq s'$ (recall that states are sets of poses). Thus, we can set $u(s) := \max(u(s), u(s'))$ for selected states $s, s' \in S$ with $s \supset s'$. If the u-values are admissible before the update, they remain admissible afterwards. The assignment could be done immediately before the local search space is generated on Line 3 in Figure 4. A stronger generalization capability might be achievable by combining function approximators with real-time heuristic search methods (similar to what is done in reinforcement learning).

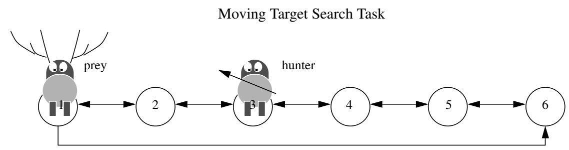
We assumed that there was no actuator and sensor uncertainty for the robot-navigation tasks. The nondeterminism resulted exclusively from missing prior knowledge. Min-Max LRTA* can also deal with a certain amount of actuator and sensor uncertainty as long as the state space remains safely explorable, that is, as long as Min-Max LRTA* is able to reach a goal state no matter which actions it has executed in the past and what the behavior of nature is. This assumption might be too pessimistic and make planning tasks wrongly appear to be unsolvable.

If the minimax goal distance of the start state is finite but the domain is not safely explorable, this problem can sometimes be addressed by increasing the local search spaces sufficiently. In this case, Min-Max LRTA* has to guarantee that it does not execute actions that can result in states with infinite minimax goal distances, that is, actions whose effect cannot necessarily be undone. If Min-Max LRTA* always determines a plan for goal-directed navigation tasks after whose execution the belief state is guaranteed to contain either only goal poses, only poses in the current belief, or only poses in the start belief, 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 [41]. Thus, Min-Max LRTA* avoids the execution of actions that make the planning task unsolvable.

If the minimax goal distance of the start state is infinite, then the planning task cannot be solved directly with minimax search because there is no solution in the worst case – a vicious nature could trap the agent no matter which actions the agent executes. Researchers have studied different ways of avoiding the problem within a minimax search framework. [16] uses discounting, [12] makes assumptions about the outcomes of action executions, and [38] splits states. A completely different way of avoiding the problem is to give up the minimax framework and make different assumptions about the behavior of nature. Min-Max LRTA* can be changed to accommodate the assumption that nature selects the successor state according to a given probability distribution that depends only on the current states and the executed actions (rather than being an opponent). In this case, it models planning tasks as Markov decision process (MDP) models (rather than Markov games) and uses average-case (rather than worst-case) search to attempt to minimize the average (rather than worst-case) plan-execution time. This version of Min-Max LRTA* is very similar to Trial-Based Real-Time Dynamic Programming (RTDP) [1, 2] and thus also many reinforcement-learning methods. The u-values of RTDP approximate the average (rather than minimax) goal distances of the states and RTDP uses the average (rather than worst-case) u-value over all successor states in its action-selection and value-update steps in Figure 3. This enables RTDP to converge to a behavior that minimizes the average (rather than worst-case) plan-execution time. If one wants to minimize the average plan-execution time for the robot-navigation tasks, one can model them with partially observable Markov decision process (POMDP) models [22], that can also model sensor and actuator uncertainty easily. Robots that use POMDPs for navigation have recently been shown to achieve a very reliable navigation behavior in unconstrained environments, [51, 9, 27, 36, 55, 29], although POMDPs often make unjustified assumptions about the behavior of nature, such as unjustified independence assumptions. Only small POMDPs can be solved with current methods [15, 8, 34]. In theory, one could solve the POMDPs also with RTDP-BEL [6], an application of RTDP to the discretized belief space of POMDPs [6]. This corresponds to how we solved the robot-navigation tasks with Min-Max LRTA*, except that the states now correspond to probability distributions over the poses (rather than sets of poses) and thus the state space is infinite and continuous (rather than finite and discrete), and needs to get discretized. Consequently, there are trade-offs when switching from a worst-case to an average-case assumption. An average-case assumption makes a larger number of planning tasks solvable and results in a more common planning objective but often makes unjustified assumptions about the behavior of nature, requires current real-time heuristic search methods to discretize the state space which results in discretization errors and substantial increases in planning time, and makes it harder to obtain analytical results about their behavior. It is therefore important that our analytical results about properties of Min-Max LRTA*, although they do not apply to real-time heuristic search methods with the average-case assumption, at least suggest properties of these real-time heuristic search methods, for example, that they solve state spaces where all states are clustered around the goal states more easily than state spaces that do not possess this property. That our analysis can suggest properties of these real-time heuristic search methods is important because their analysis is typically avoided. Our analytical results might therefore provide first stepping stones towards a better understanding of these real-time heuristic search methods.

7 Another Application Example: Moving-Target Search

In this section, we briefly study another application of Min-Max LRTA*, namely *moving-target search*, the task being for a hunter to catch an independently moving prey. We discuss the variant of moving-target search where both agents move on a known directed graph. The hunter moves first, then they alternate moves to adjacent vertices. Both agents can always sense the current vertex of themselves and the other agent, but the hunter does not know in advance where the prey moves. The hunter catches the prey if both agents occupy the



Corresponding Non-Deterministic Single-Agent Domain

state = (hunter, prey)	action of hunter	successor states
(done) = goal state	—	—
(3,1) = start state	move to 2	(2,6) or (done)
	move to 4	(4,2) or (4,6)
(2,6)	move to 1	(1,5)
	move to 3	(3,5)
(4,2)	move to 3	(3,1) or (done)
	move to 5	(5,1) or (5,3)
(4,6)	move to 3	(3,5)
	move to 5	(done)
...	...	...

Figure 22: Moving-Target Search Task

same vertex. In our framework, the agent is the hunter. It is straightforward to map the (two-agent) moving-target search task to a (single-agent) planning task against nature in a nondeterministic domain (Figure 22). The hunter can catch the prey for sure if the minimax goal distance of the start state in the nondeterministic domain is finite. Min-Max LRTA* can be used to determine a behavior for the hunter that catches the prey if the nondeterministic domain is safely explorable.

This application of Min-Max LRTA* is interesting since it makes different assumptions than other real-time heuristic search methods that have been used for moving-target search. Min-Max LRTA* can solve moving-target search tasks that these search methods cannot solve. In particular, the *Moving-Target Search method* (MTS) [21] is another real-time heuristic search method that applies to moving-target search, but utilizes LRTA* for the hunter in a different way. It learns the following behavior for the hunter: always move to an adjacent vertex that is on a shortest path to the current vertex of the prey. Ishida and Korf [21] prove that the hunter eventually catches the prey on a strongly connected graph if it is faster moving than the prey. Notice the difference between the two real-time heuristic search methods: Obviously, one has to make some assumptions to ensure that the prey cannot force the hunter into a cycle in which the hunter cannot decrease its distance to the prey. Min-Max LRTA* assumes that the nondeterministic domain is safely explorable, which means that a skillful hunter is able to corner the prey. MTS, on the other hand, does not restrict the topology of the graph, but has to assume that the hunter has a speed advantage over the prey. Consider, for example, the graph from Figure 22 (notice that one of its edges is directed) and assume that both agents are equally fast moving. Fully informed MTS always follows the prey at the same distance if the prey runs around in an anti-clockwise cycle. Min-Max LRTA*, however, eventually moves left until the prey takes the one-way street and later moves right until the prey is caught, no matter how the prey behaves.

8 Conclusions

Classical planners traditionally assume that domains are deterministic, and the few planners that can operate in nondeterministic domains often assume that the state of the world is completely observable. We, on the other hand, studied planning tasks in nondeterministic domains, including domains where the state of the world cannot be observed completely, and presented both planning methods and first analytical results about their behavior. The analytical results are important because planning in nondeterministic domains is still poorly understood. Our results provide some insight into the behavior of planners in nondeterministic domains and might also be useful as examples for how to obtain analytical results.

We applied real-time heuristic search methods to planning tasks in nondeterministic domains. We demonstrated that they can be used to speed up planning in these domains and analyzed their behavior. Our real-time heuristic search method, Min-Max LRTA*, extends LRTA* to nondeterministic domains by interleaving minimax searches and plan executions. It is guaranteed to solve all planning tasks in safely explorable domains, that is, if the minimax goal distance of every state is finite. Min-Max LRTA* has the following advantages: First, different from the many existing ad-hoc planning methods that interleave planning and plan executions, it has a solid theoretical foundation and is domain independent. Second, it allows for fine-grained control over how much planning to do between plan executions. Third, it can use heuristic knowledge (in the form of admissible initial u-values) to guide planning which can reduce the planning time without increasing the plan-execution time. Fourth, it can be interrupted at any state and resume execution at a different state. In other words, other control programs can take over control at arbitrary times if desired. Fifth, it amortizes learning over several planning episodes, which allows it to find a plan with a suboptimal plan-execution time fast and then improve the plan-execution time as it solves similar planning tasks, until the plan-execution time is satisficing or optimal. Thus, it can always have a small sum of planning and plan-execution time and still asymptotically minimize the plan-execution time in the long run in case similar planning tasks unexpectedly repeat. This is an important property since no planning method that executes actions before it has found a complete plan can guarantee a good plan-execution time right away, and planning methods that do not improve their plan-execution time do not behave efficiently in the long run in case similar planning tasks unexpectedly repeat. Since Min-Max LRTA* partially relies on observing the actual successor states of action executions, it does not plan for all possible successor states and thus can still have computational advantages even over several search episodes compared to a complete minimax search if nature is not as malicious as a minimax search assumes and some successor states do not occur in practice.

Min-Max LRTA* reduces to LRTA* in deterministic domains. Thus, our analytical results about the properties of Min-Max LRTA* also apply to LRTA* and provide new insight into the behavior of this popular real-time heuristic search method. Min-Max LRTA* assumes that the execution of an action always results in the worst successor state for the agent. Min-Max LRTA* can be changed to assume that the execution of an action results in a successor state that is determined by a probability distribution that depends only on the action and the state it is executed in and then becomes very similar to Trial-Based Real-Time Dynamic Programming and thus also many reinforcement-learning methods. While our analytical results do not apply to this case, they still suggest properties of the resulting real-time heuristic search method and provide first stepping stones towards more powerful and complex real-time heuristic search methods.

We applied Min-Max LRTA* to simulated robot-navigation tasks in mazes, where the robots know the maze but do not know their initial position and orientation, and showed experimentally that Min-Max LRTA* solves the robot-navigation tasks fast, converges quickly, and requires only a small amount of memory. We also demonstrated that Min-Max LRTA* generalizes existing solution methods for these tasks such as the Information-Gain Method, that has been used on actual robots. It does so by utilizing heuristic knowledge, allowing for fine-grained control over how much planning to do between plan-executions, and enabling robots to improve their plan-execution time as they solve similar planning tasks. Finally, we discussed how Min-Max LRTA* can be applied to moving-target search, the task of catching a moving target.

Acknowledgments

Thanks to Matthias Heger, Richard Korf, Michael Littman, Tom Mitchell, Illah Nourbakhsh, Patrawadee Prasangsit, and Reid Simmons for helpful discussions. Special thanks to Richard Korf, Michael Littman, and Reid Simmons for their extensive comments on this work and to Joseph Pemberton for making his maze generation program available to us. The Intelligent Decision-Making Group is partly supported by an NSF Award to Sven Koenig under contract IIS-9984827. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations and agencies or the U.S. government.

A Appendix: The Proofs

In this appendix, we prove the two theorems that we did not prove in the main text. First, we prove Theorem 1 and then Theorem 2. Throughout the appendix, the time superscript t refers to the values of the variables immediately before the $(t+1)$st value-update step (minimax search) of Min-Max LRTA* (Line 4 in Figure 4) without Line 8.

A.1 Proof of Theorem 1

We prove the following version of Theorem 1.

Theorem 7 *For all times $t = 0, 1, 2, \dots$ (until termination): Assume that $u^t(s) \in [0, \infty]$ for all $s \in S$, and define $u^{t+1}_{opt}(s) \in [0, \infty]$ as*

$$u^{t+1}_{opt}(s) = \begin{cases} u^t(s) & \text{if } s \notin S^t_{lss} \\ \max(u^t(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} u^{t+1}_{opt}(s')) & \text{otherwise} \end{cases} \qquad \text{for all } s \in S.$$

Then, the $(t+1)$st minimax search terminates with $u^{t+1}(s) = u^{t+1}_{opt}(s)$ for all $s \in S$.

Proof: We first prove that the minimax search terminates and then that it determines the correct u-values.

The minimax search terminates: It terminates if the u-values of all states in the local search space are smaller than infinity (Line 2). Otherwise, it either changes the u-value of another state from infinity to some other value (Line 5) or, if that is not possible, terminates (Line 4). Thus, it terminates eventually.

The minimax search determines the correct u-values: Consider the $(t+1)$st execution of the minimax-search method for any time $t = 0, 1, 2, \dots$ (until termination). The u-values $u^{t+1}(s)$ are correct for all states s that do not belong to the local search space S^t_{lss} because they do not change during the minimax search and thus $u^{t+1}(s) = u^t(s) = u^{t+1}_{opt}(s)$. To show that the u-values $u^{t+1}(s)$ are also correct for all states of the local search space consider any time during the minimax search. Then, $u(s) = u^{t+1}_{opt}(s)$ for all $s \in S^t_{lss}$ with $u(s) < \infty$, as we prove below. It follows that, after the minimax search, the u-values are correct for all states of the local search space whose u-values are smaller than infinity. To show that the u-values are also correct for all states of the local search space whose u-values equal infinity, suppose that this is not the case. Then, the minimax search terminates on Line 4 and there are states in the local search space whose u-values are, incorrectly, infinity. Among all states in the local search space whose u-values are infinity, consider a state s with the minimal value $u^{t+1}_{opt}(s)$, any action $a :=$ one-of $\arg\min_{a \in A(s)} \max_{s' \in succ(s,a)} u^{t+1}_{opt}(s')$, and any state $s' \in succ(s, a)$. Then, $u^{t+1}(s) = \infty$ and $u^{t+1}_{opt}(s) < \infty$, that is, the u-value of s is, incorrectly, infinity. Since $u^{t+1}_{opt}(s') < u^{t+1}_{opt}(s) < \infty$, it holds that either $s' \in S \setminus S^t_{lss}$, which implies $u(s') = u^t(s') = u^{t+1}_{opt}(s') < \infty$, or $s' \in S^t_{lss}$ with $u(s') < \infty$ according to our choice of s. Also, $u^t(s) < \infty$ since $u^{t+1}_{opt}(s) < \infty$. Then, however, the minimax search could not have terminated on Line 4 because $\max(u^t(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} u(s')) < \infty$, which is a contradiction. Thus, the minimax search determines the correct u-values.

We prove by induction that, at any time during the $(t+1)$st execution of the minimax-search method, $u(s) = u^{t+1}_{opt}(s)$ for all $s \in S^t_{lss}$ with $u(s) < \infty$. This holds initially since $u(s) = \infty$ for all $s \in S^t_{lss}$. Now suppose that the induction hypothesis holds when an $\overline{s} \in S^t_{lss}$ is chosen on Line 3 and let $u(s)$ denote the u-values at this point in time. Suppose further that the subsequent assignment on Line 5 results in $u_{new}(\overline{s}) \neq u^{t+1}_{opt}(\overline{s})$. In general, $u(s') \geq u^{t+1}_{opt}(s')$ for all $s' \in S^t_{lss}$ since either $u(s') = \infty$ or $u(s') = u^{t+1}_{opt}(s')$ according to the induction hypothesis. Then,

$$u_{new}(\overline{s}) \quad = \quad \max(u^t(\overline{s}), 1 + \min_{a \in A(\overline{s})} \max_{s' \in succ(\overline{s},a)} u(s'))$$

$$
\begin{aligned}
&\geq \max(u^t(\overline{s}), 1 + \min_{a \in A(\overline{s})} \max_{s' \in succ(\overline{s},a)} u^{t+1}_{opt}(s')) \\
&= u^{t+1}_{opt}(\overline{s}).
\end{aligned}
$$

Since $u_{new}(\overline{s}) \stackrel{\text{Assumption}}{\neq} u^{t+1}_{opt}(\overline{s})$, it holds that $u_{new}(\overline{s}) = \max(u^t(\overline{s}), 1 + \min_{a \in A(\overline{s})} \max_{s' \in succ(\overline{s},a)} u(s')) > u^{t+1}_{opt}(\overline{s})$. Among all states in the local search space whose u-values are infinity, consider a state s with the minimal value $u^{t+1}_{opt}(s)$, any action a := one-of $\arg\min_{a \in A(s)} \max_{s' \in succ(s,a)} u^{t+1}_{opt}(s')$, and any state $s' \in succ(s, a)$. Since $u^{t+1}_{opt}(s') < u^{t+1}_{opt}(s) \leq u^{t+1}_{opt}(\overline{s}) < u_{new}(\overline{s}) < \infty$, it holds that either $s' \in S \setminus S^t_{lss}$ or $s' \in S^t_{lss}$ with $u(s') < \infty$ according to our choice of s. In either case, $u^{t+1}_{opt}(s') = u(s')$ according to the definition of $u^{t+1}_{opt}(s')$ or the induction hypothesis. Then,

$$
\begin{aligned}
&\max(u^t(\overline{s}), 1 + \min_{a \in A(\overline{s})} \max_{s' \in succ(\overline{s},a)} u(s')) \\
&\quad > u^{t+1}_{opt}(\overline{s}) \\
&\quad \geq u^{t+1}_{opt}(s) \\
&\quad = \max(u^t(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} u^{t+1}_{opt}(s')) \\
&\quad \geq \max(u^t(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} u(s')).
\end{aligned}
$$

But then the minimax search could not have chosen $\overline{s}$. This is a contradiction. Thus, $u_{new}(\overline{s}) = u^{t+1}_{opt}(\overline{s})$. After $\overline{s}$ has been assigned this value on Line 5, it cannot be assigned another value later, since $u_{new}(\overline{s}) < \infty$. ■

A.2 Proof of Theorem 2

Theorem 8 *(= Theorem 2) Admissible initial u-values remain admissible after every value-update step of Min-Max LRTA* and are monotonically nondecreasing.*

Proof: The u-values are monotonically nondecreasing because either they do not change or, according to Line 5 of the minimax-search method, $u^{t+1}(s) = \max(u^t(s), \ldots) \geq u^t(s)$. The intuitive reason why initially admissible u-values remain admissible is the same as the reason given on page 2 for LRTA*. In the following, we formalize that reason for Min-Max LRTA*: Assume that the u-values $u^t(s)$ are admissible. Then, for all $s \in S \setminus S^t_{lss}$, $gd(s) \stackrel{\text{Admissibility}}{\geq} u^t(s) = u^{t+1}(s) = u^t(s) \stackrel{\text{Admissibility}}{\geq} 0$. Furthermore, for all $s \in S^t_{lss}$, $gd(s) \geq u^{t+1}(s)$ as we show below. It follows that, for all $s \in S^t_{lss}$, $gd(s) \geq u^{t+1}(s) \stackrel{\text{Monotonicity}}{\geq} u^t(s) \stackrel{\text{Admissibility}}{\geq} 0$. Therefore, the u-values $u^{t+1}(s)$ are admissible as well.

We prove by induction that, for all $s \in S^t_{lss}$, $gd(s) \geq u^{t+1}(s)$. Consider an ordering s_i for $i = 1, 2, \ldots, |S^t_{lss}|$ of all $s \in S^t_{lss}$ according to their increasing minimax goal distance. We show by induction on i that $gd(s_i) \geq u^{t+1}(s_i)$. We consider two cases: First, $gd(s_i) = \infty$. Then, it holds trivially that $gd(s_i) \geq u^{t+1}(s_i)$. Second, $gd(s_i) < \infty$. Then, $s_i \in S^t_{lss}$ implies $s_i \in S \setminus G$ per definition of S^t_{lss}. Thus, $gd(s_i) = 1 + \min_{a \in A(s_i)} \max_{s' \in succ(s_i,a)} gd(s')$. Let a' := one-of $\arg\min_{a \in A(s_i)} \max_{s' \in succ(s_i,a)} gd(s')$. Then, $gd(s_i) = 1 + \max_{s' \in succ(s_i,a')} gd(s')$ and thus $gd(s') < gd(s_i)$ for all $s' \in succ(s_i, a')$. In general, $gd(s') \geq u^{t+1}(s')$ for all $s' \in succ(s_i, a')$ since either $s' \in S \setminus S^t_{lss}$ (see above) or $s' = s_j$ with $j < i$ per induction hypothesis. Then,

$$
\begin{aligned}
gd(s_i) &= \max(gd(s_i), gd(s_i)) \\
&= \max(gd(s_i), 1 + \max_{s' \in succ(s_i,a')} gd(s')) \\
&\geq \max(gd(s_i), 1 + \max_{s' \in succ(s_i,a')} u^{t+1}(s'))
\end{aligned}
$$

$$
\begin{aligned}
&\geq && \max(gd(s_i), 1 + \min_{a \in A(s_i)} \max_{s' \in succ(s_i,a)} u^{t+1}(s')) \\
&\overset{\text{Admissibility}}{\geq} && \max(u^t(s_i), 1 + \min_{a \in A(s_i)} \max_{s' \in succ(s_i,a)} u^{t+1}(s')) \\
&\overset{\text{Theorem 1}}{=} && u^{t+1}(s_i). \blacksquare
\end{aligned}
$$

References

[1] A. Barto, S. Bradtke, and S. Singh. Real-time learning and control using asynchronous dynamic programming. Technical Report 91–57, Department of Computer Science, University of Massachusetts at Amherst, Amherst (Massachusetts), 1991.

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

[3] A. Barto, R. Sutton, and C. Watkins. Learning and sequential decision making. Technical Report 89–95, Department of Computer Science, University of Massachusetts at Amherst, Amherst (Massachusetts), 1989.

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

[5] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence – Special Issue on Heuristic Search*, 2000. (Accepted).

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

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

[8] A. Cassandra. *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Department of Computer Science, Brown University, Providence (Rhode Island), 1997.

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

[10] A. Cimatti and M. Roveri. Conformant planning via model checking. In *Proceedings of the European Conference on Planning*, 1999.

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

[12] N. Friedman and D. Koller. Qualitative planning under assumptions: A preliminary report. In *Proceedings of the AAAI Spring Symposium on Decision-Theoretic Planning*, pages 106–112, 1994. Available as AAAI Technical Report SS-94-06.

[13] M. Genesereth and I. Nourbakhsh. Time-saving tips for problem solving with incomplete information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 724–730, 1993.

[14] F. Giunchiglia and P. Traverso. Planning as model checking. In *Proceedings of the European Conference on Planning*, 1999.

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

[16] M. Heger. The loss from imperfect value functions in expectation-based and minimax-based tasks. *Machine Learning*, 22(1–3):197–225, 1996.

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

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

[19] T. Ishida. Two is not always better than one: Experiences in real-time bidirectional search. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 185–192, 1995.

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

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

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

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

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

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

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

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

[28] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

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

[30] R. Korf. Real-time heuristic search: First results. In *Proceedings of the National Conference on Artificial Intelligence*, pages 133–138, 1987.

[31] R. Korf. Real-time heuristic search: New results. In *Proceedings of the National Conference on Artificial Intelligence*, pages 139–144, 1988.

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

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

[34] M. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Department of Computer Science, Brown University, Providence (Rhode Island), 1996. Available as Technical Report CS-96-09.

[35] T. Lozano-Perez, M. Mason, and R. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[36] S. Mahadevan, G. Theocharous, and N. Khaleeli. Rapid concept learning for mobile robots. *Autonomous Robots Journal*, 5:239–251, 1998.

[37] S. Matsubara and T. Ishida. Real-time planning by interleaving real-time search with subgoaling. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 122–127, 1994.

[38] A. Moore and C. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21(3):199–233, 1995.

[39] I. Nourbakhsh. *Interleaving Planning and Execution*. PhD thesis, Department of Computer Science, Stanford University, Stanford (California), 1996.

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

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

[42] I. Nourbakhsh and M. Genesereth. Assumptive planning and execution: a simple, working robot architecture. *Autonomous Robots Journal*, 3(1):49–67, 1996.

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

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

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

[46] S. Russell and P. Norvig. *Artificial Intelligence – A Modern Approach*. Prentice Hall, 1995.

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

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

[49] R. Schapire. *The Design and Analysis of Efficient Learning Algorithms*. MIT Press, 1992.

[50] B. Selman. Stochastic search. In *MIT Encyclopedia of Cognitive Science*, 1998.

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

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

[53] C. Szepesvári. Learning and exploitation do not conflict under minimax optimality. In *Proceedings of the European Conference on Machine Learning*, pages 242–249, 1997.

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

[55] S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31(5):1–25, 1998.

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

[57] S. Whitehead. *Reinforcement Learning for the Adaptive Control of Perception and Action*. PhD thesis, Department of Computer Science, University of Rochester, Rochester (New York), 1992.

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