Solving Robot Navigation Problems with Initial Pose Uncertainty Using Real-Time Heuristic Search*

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

Reid G. Simmons
Computer Science Department
Carnegie Mellon University
reids@cs.cmu.edu

Abstract

We study goal-directed navigation tasks in mazes, where the robots know the maze but do not know their initial pose (position and orientation). These search tasks can be modeled as planning tasks in large non-deterministic domains whose states are sets of poses. They can be solved efficiently by interleaving planning and plan execution, which can reduce the sum of planning and plan-execution time because it allows the robots to gather information early. We show how Min-Max LRTA*, a real-time heuristic search method, can solve these and other planning tasks in non-deterministic domains efficiently. It allows for fine-grained control over how much planning to do between plan executions, uses heuristic knowledge to guide planning, and improves its plan-execution time as it solves similar planning tasks, until its plan-execution time is at least worst-case optimal. We also show that Min-Max LRTA* solves the goal-directed navigation tasks fast, converges quickly, and requires only a small amount of memory.

Introduction

Situated agents (such as robots) have to take their planning time into account to solve planning tasks efficiently (Good 1971). For single-instance planning tasks, for example, they should attempt to minimize the sum of planning and plan-execution time. Finding plans that minimize the plan-execution time is often intractable. Interleaving planning and plan execution is a general principle that can reduce the planning time and thus also the sum of planning and plan execution time for sufficiently fast agents in non-deterministic domains (Genesereth & Nourbakhsh 1993). Without interleaving planning and plan execution, the agents have to find a large conditional plan that solves the planning task. When interleaving planning and plan execution, 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 we consider goal-directed navigation tasks in mazes where the robots know the maze but do not know their initial pose (position and orientation). Interleaving planning and plan execution 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.

Agents that interleave planning and plan execution 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 LRTA*, a real-time heuristic search method that extends LRTA* (Korf 1990) to non-deterministic domains, can be used to address these problems. Min-Max LRTA* interleaves planning and plan execution and plans 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 them in their current situation. It allows for fine-grained control over how much planning to do between plan executions, uses heuristic knowledge to guide planning, and improves its plan-execution time as it solves similar planning tasks, until its plan-execution time is at least worst-case optimal. We show that Min-Max LRTA* solves the goal-directed navigation tasks fast, converges quickly, and requires only a small amount of memory.

Planning methods that interleave planning and plan execution have been studied before. This includes assumptive planning, deliberation scheduling (including anytime algorithms), on-line algorithms and competitive analysis, real-time heuristic search, reinforcement learning, robot exploration techniques, and sensor-based planning. The proceedings of the AAAI-97 Workshop on On-Line Search give a good overview of these techniques. The contri-

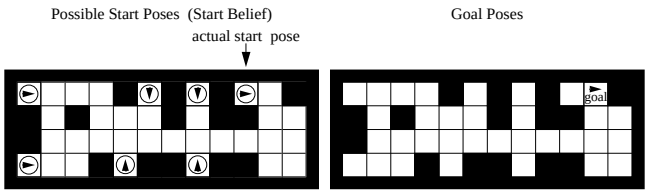

Figure 1: Goal-Directed Navigation Task #1

butions of this paper are threefold: First, this paper extends our Min-Max LRTA* (Koenig & Simmons 1995), a real-time heuristic search method for non-deterministic domains, from look-ahead one to arbitrary look-aheads. It shows how the search space of Min-Max LRTA* can be represented more compactly than is possible with minimax trees and discusses how it can be solved efficiently using methods from Markov game theory. Second, this paper illustrates an advantage of real-time heuristic search methods that has not been studied before. It was already known that real-time heuristic search methods are efficient alternatives to traditional search methods 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 (Korf 1993). They have also been used to solve STRIPS-type planning tasks (Bonet, Loerincs, & Geffner 1997). The reason why they are efficient in deterministic domains is because they trade-off minimizing planning time and minimizing plan-execution time. However, many domains from robotics, control, and scheduling are nondeterministic. We demonstrate that real-time heuristic search methods can also be used to speed up problem solving in nondeterministic domains and that there is an additional reason why they are efficient in these domains, namely that they 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. Third, this paper demonstrates an advantage of Min-Max LRTA* over most previous planning methods that interleave planning and plan execution, namely that it improves its plan-execution time as it solves similar planning tasks. As recognized in both (Dean *et al.* 1995) and (Stentz 1995), 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, and methods that do not improve their plan-execution time do not behave efficiently in the long run in case similar planning tasks unexpectedly repeat.

The Robot Navigation Problem

We study goal-directed navigation tasks with initial pose uncertainty (Nourbakhsh 1996), an example of which is shown 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). We assume that there is no uncertainty in actuation and sensing. (In our mazes, it is indeed possible to make actuation and sensing approximately 100 percent reliable.) 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. The task of the robot is 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 the goal-directed navigation task includes localizing the robot sufficiently so that it knows that it is at a goal pose when it stops. 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 $P_{start} = \{p : p \in P \wedge o(p) = o(p_{start})\}$. The robot has to navigate to any pose in $P_{goal} \subseteq P$ and stop.

Even if the robot is not certain about its pose, it can 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 ("belief"). For example, if the 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, then the start belief of the robot contains the seven possible start poses shown in the figure. We use the following notation: B is the set of beliefs, b_{start} the start belief, and B_{goal} the set of goal beliefs. $A(b)$ 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,

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

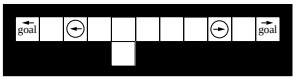

Figure 2: Goal-Directed Navigation Task #2

$$\begin{aligned} A(b) &= A(p) \text{ for any } p \in b \\ O(b,a) &= \{o(succ(p,a)) : p \in b\} \\ succ(b,a,o) &= \{succ(p,a) : p \in b \wedge o(succ(p,a)) = o\} \end{aligned}$$

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. To understand the definition of B_{goal}, 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. Figure 2 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. If it is important that the robot knows which goal pose it is in, we define $B_{goal} = \{b : b \subseteq P_{goal} \wedge |b| = 1\}$. Min-Max LRTA* can also be applied to localization tasks. These tasks are identical to the goal-directed navigation tasks except that the robot has to achieve only certainty about its pose. In this case, we define $B_{goal} = \{b : b \subseteq P \wedge |b| = 1\}$.

The robot navigation domain is deterministic and small (*pose space*). However, the beliefs of the robot depend on its observations, which the robot cannot predict with certainty since it is uncertain about its pose. We therefore formulate the goal-directed navigation tasks as planning tasks in a 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. After an action execution, the robot will usually not be able to determine its current pose with certainty, but it can always determine its current belief for sure. We use the following notation: S denotes the finite set of states of the domain, $s_{start} \in S$ the start state, and $G \subseteq S$ the set of goal states. $A(s)$ is the finite set of (potentially nondeterministic) actions that can be executed in state $s \in S$. $succ(s,a)$ denotes the set of successor states that can result from the execution of action $a \in A(s)$ in state $s \in S$. Then,

$$\begin{aligned} S &= B \\ s_{start} &= b_{start} \\ G &= B_{goal} \\ A(s) &= A(b) \text{ for } s = b \end{aligned}$$

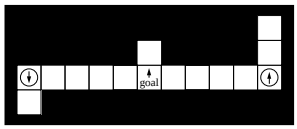

Figure 3: Goal-Directed Navigation Task #3

$$succ(s,a) = \{succ(b,a,o) : o \in O(b,a)\} \text{ for } s = b$$

The actual successor state that results from the execution of action a in state $s = b$ is $succ(b,a,o)$ if observation o is made after the action execution. In general, traversing any path from the start state in the belief space to a goal state solves the goal-directed navigation task. To make all goal-directed navigation tasks solvable we require the mazes to be strongly connected (every pose can be reached from every other pose) and asymmetrical (localization is possible). This modest assumption allows the robot to solve a goal-directed navigation task by first localizing itself and then moving to a goal pose, although this behavior often does not minimize the plan-execution time. Figure 3 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.

Min-Max LRTA*

Min-Max Learning Real-Time A* (Min-Max LRTA*) is a real-time heuristic search method that extends LRTA* (Korf 1990) to nondeterministic domains by interleaving minimax search in local search spaces and plan execution (Figure 4). Similar to game-playing approaches and reinforcement-learning methods such as $\hat{Q}$-Learning (Heger 1996), 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. Acting in deterministic domains is then simply a special case where every action uniquely determines the successor state. Min-Max LRTA* uses minimax search to solve planning tasks in nondeterministic domains, a worst-case search method that 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 Min-Max LRTA* as far away from the goal state as possible. 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 *u-value* $u(s) \geq 0$ with each state $s \in S$.

Initially, the u-values $u(s) \geq 0$ are approximations of the minimax goal distances (measured in action executions) for all $s \in S$.

Given a set X, the expression "one-of X" returns an element of X according to an arbitrary rule. A subsequent invocation of "one-of X" can return the same or a different element. The expression "arg $\min_{x \in X} f(x)$" returns the set $\{x \in X : f(x) = \min_{x' \in X} f(x')\}$.

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: Min-Max LRTA*

The u-values approximate the minimax goal distances of the states. 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. Using action executions to measure the minimax goal distances and thus plan-execution times is reasonable if every action can be executed in about the same amount of time. The minimax goal distances are defined by the following set of equations for all $s \in S$:

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

Min-Max LRTA* updates the u-values as the search progresses and uses them to determine which actions to execute. It 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). 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. It then updates the u-values of all states in the local search space (Line 4) and, based on these u-values, 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.

Min-Max LRTA* uses minimax search to update the u-values in the local search space (Figure 5). 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

search method assigns each state its minimax goal distance under the assumption that the u-values of all states in the local search space are lower bounds on the correct minimax goal distances and that the u-values of all states outside of the local search space correspond to their correct minimax goal distances. Formally, if $u(s) \in [0, \infty]$ denotes the u-values before the minimax search and $\bar{u}(s) \in [0, \infty]$ denotes the u-values afterwards, then $\bar{u}(s) = \max(u(s), 1 + \min_{a \in A(s)} \max_{s' \in succ(s,a)} \bar{u}(s'))$ for all $s \in S_{lss}$ and $\bar{u}(s) = u(s)$ otherwise. Min-Max LRTA* could represent the local search space as a minimax tree, 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*), which can be a problem for goal-directed navigation tasks in large empty spaces. Since the number of different states often grows only polynomially in the depth of the tree, Min-Max LRTA* represents the local search space more compactly as a graph that contains 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 simple method that is related to more general dynamic programming methods from Markov game theory (Littman 1996). It updates all states in the local search space in the order of their increasing new u-values. This ensures that the u-value of each state is updated only once. More details on the minimax search method are given in (Koenig 1997).

After Min-Max LRTA* has updated the u-values, it greedily chooses the action for execution that minimizes the u-value of the successor state in the worst case (ties are broken arbitrarily). The rationale behind this 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* can either generate another local search space, update the u-values of all states that it contains, and select another ac-

tion for execution. However, 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* also has the option (Line 8) to select another action for execution based on the current u-values. 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 $\{s\}$. Since this does not change the u-values, Min-Max LRTA* can, in this case, also skip the minimax search. In the experiments, we use Min-Max LRTA* with Line 8, because it utilizes more information of the searches in the local search spaces.

Features of Min-Max LRTA*

An 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. This is an advantage of Min-Max LRTA* over Trial-Based Real-Time Dynamic Programming (Barto, Bradtke, & Singh 1995), another generalization of LRTA* to nondeterministic domains. Trial-Based Real-Time Dynamic Programming assumes that nature chooses successor states randomly according to given probabilities. Therefore, it does not apply to planning in the pose space when it is impossible to make reliable assumptions about the behavior of nature, including tasks with perceptual aliasing (Chrisman 1992) and the goal-directed navigation tasks studied in this paper. A disadvantage of Min-Max LRTA* is that it cannot solve all planning tasks. This is so because it interleaves minimax searches and plan execution. 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 becomes unsolvable. Despite these disadvantages, Min-Max LRTA* is guaranteed to solve all safely explorable domains. These are domains for which 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. For example, the belief space of the robot navigation domain is safely explorable according to our assumptions, and so are other nondeterministic domains from robotics, including manipulation and assembly domains, which explains why using minimax methods is popular in robotics (Lozano-Perez, Mason, & Taylor 1984). We therefore assume in the following that Min-Max LRTA* is applied to safely explorable domains. For similar reasons, we also assume in the following that action executions cannot leave the current state unchanged. The class of domains that Min-Max LRTA* is guaranteed to solve could be extended further by using the methods in (Nourbakhsh 1996) to construct the local search spaces.

Min-Max LRTA* has three key features, namely that it allows for fine-grained control over how much planning to do between plan executions, uses heuristic knowledge to guide planning, 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 method that executes actions before it knows their complete consequences can guarantee a good plan-execution time right away. In the following, we explain these features of Min-Max LRTA* in detail.

Heuristic Knowledge

Min-Max LRTA* uses heuristic knowledge to guide planning. This knowledge is provided in the form of admissible initial u-values. U-values are admissible if and only if $0 \leq u(s) \leq gd(s)$ for all $s \in S$. In deterministic domains, this definition reduces to the traditional definition of admissible heuristic values for A* search (Pearl 1985). The larger its initial u-values, the better informed Min-Max LRTA*.

Theorem 1 *Let $\hat{u}(s)$ denote the initial u-values. Then, Min-Max LRTA* with initially admissible u-values reaches a goal state after at most $\hat{u}(s_{start}) + \sum_{s \in S}[gd(s) - \hat{u}(s)]$ action executions, regardless of the behavior of nature.*

All proofs can be found in (Koenig 1997). The theorem shows that Min-Max LRTA* with initially admissible u-values reaches a goal state in safely explorable domains after a finite number of action executions, that is, it is correct. The larger the initial u-values, the smaller the upper bound on the number of action executions and thus the smaller its plan-execution time. For example, Min-Max LRTA* is fully informed if the initial u-values equal the minimax goal distances of the states. In this case, Theorem 1 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 method can do better in the worst-case. For the 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 execution, 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 the localization tasks, on the other hand, it is difficult to obtain better informed initial u-values than those provided by the zero heuristic (zero-initialized u-values).

Fine-Grained Control

Min-Max LRTA* allows for fine-grained control over how much planning to do between plan executions. For example, Min-Max LRTA* with line 8 and $S_{lss} = S \setminus G = S \cap \overline{G}$ performs a complete minimax search without interleaving planning and plan execution, 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 look-aheads benefit agents that can execute plans with a similar speed as they can generate them, because they prevent these agents from being idle, which can minimize the sum of planning and plan-execution time if the heuristic knowledge guides the search sufficiently. Larger look-aheads are needed for slower agents, such as robots.

Improvement of Plan-Execution Time

If Min-Max LRTA* solves the same planning task repeatedly (even with different start states) it can improve its behavior over time by transferring domain knowledge, in the form of u-values, between planning tasks. This is a familiar concept: One can argue that the minimax searches that Min-Max LRTA* performs between plan executions are independent of one another and that they are connected only via the u-values that transfer domain knowledge between them. To see why Min-Max LRTA* can do the same thing between planning tasks, consider the following theorem.

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

Now assume that a series of planning tasks in the same domain with the same set of goal states are given, for example goal-directed navigation tasks with the same goal poses in the same maze. The actual start poses or the beliefs of the robot about its start poses do not need to be identical. If the initial u-values of Min-Max LRTA* are admissible for the first planning task, then they are also admissible 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 the start states. 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. 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.

Theorem 3 *Min-Max LRTA* with initially admissible u-values reaches a goal state after at most $gd(s_{start})$ action executions, regardless of the behavior of nature, if its u-values do not change during the search.*

Theorem 4 *Assume that Min-Max LRTA* maintains u-values across a series of planning tasks in the same domain with the same set of goal states. Then, the number of planning tasks for which Min-Max LRTA* with initially admissible u-values reaches a goal state after more than $gd(s_{start})$ action executions is bounded from above by a finite constant that depends only on the domain and goal states.*

Proof: If Min-Max LRTA* with initially admissible 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 3. This can happen only a finite 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. ■

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 4, 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

1. $S_{lss} := \{s\}$.
2. Improve $u(s)$ 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 6: Generating Local Search Spaces

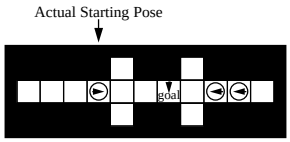

Figure 7: Goal-Directed Navigation Task #4

successor states of action executions, and nature can wait an arbitrarily long time to reveal it or choose not to reveal it at all. This can prevent the u-values from converging after a bounded number of action executions and is the reason why we analyzed its behavior 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, for example, because the actual start pose of the robot never equals the worst possible pose among all start poses that are consistent with its start belief.

Other Navigation Methods

Min-Max LRTA* is a domain-independent planning method that does not only apply to goal-directed navigation and localization tasks with initial pose uncertainty, but to planning tasks in nondeterministic domains in general. It can, for example, also be used for moving-target search, the task of a hunter to catch a moving prey (Koenig & Simmons 1995). In the following, we compare Min-Max LRTA* to two more specialized planning methods that can also be used to solve the goal-directed navigation or localization tasks. An important advantage of Min-Max LRTA* over these methods is that it can improve its plan-execution time as it solves similar planning tasks.

Information-Gain Method

The *Information-Gain Method* (IG method) (Genesereth & Nourbakhsh 1993) first demonstrated the advantage of interleaving planning and plan execution for goal-directed navigation tasks.[1] It uses breadth-first search (iterative deepening) on an and-or graph 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, the robot has either solved the goal-directed navigation task or at least reduced the number of poses it can be in. This way, the IG method guarantees progress towards the goal. There are similarities between the IG method and Min-Max LRTA*: Both methods interleave minimax searches and plan execution. Zero-initialized Min-Max LRTA* that generates the local search spaces for goal-directed navigation tasks with the method from Figure 6 exhibits a similar behavior as the IG method: 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, 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. There are also differences between the IG method and Min-Max LRTA*: Min-Max LRTA* can use small look-aheads that do not guarantee a gain in information and it can improve its plan-execution time as it solves similar planning tasks, at the cost of having to maintain u-values. The second advantage is important because no method that interleaves planning and plan execution can guarantee a good plan-execution time on the first run. For instance, consider the goal-directed navigation task from Figure 7 and assume that Min-Max LRTA* generates the local search spaces with the method from Figure 6. Then, both the IG method and zero-initialized Min-Max LRTA* move forward, because this is the fastest way to eliminate

[1] (Genesereth & Nourbakhsh 1993) 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.

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 with the same start pose, Min-Max LRTA* eventually learns this behavior.

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 (Kohavi 1978). An adaptive homing sequence is a conditional plan with the same property (Schapire 1992). For every reduced deterministic finite state automaton, there exists a homing sequence that contains at most $(n-1)^2$ actions. Finding a shortest homing sequence is NP-complete but a suboptimal homing sequence of at most $(n-1)^2$ actions can be found in polynomial time (Schapire 1992). 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.

Extensions of Min-Max LRTA*

Min-Max LRTA* uses minimax searches in the local search spaces to update its u-values. For the robot navigation tasks, it is possible to combine this with updates over a greater distance, 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. It is also straightforward to modify Min-Max LRTA* so that it does not assume that all actions can be executed in the same amount of time, and change the theorems appropriately. Finally, it is also straightforward (but not terribly exciting) to drop the assumption that action executions cannot leave the current state unchanged, and change the theorems appropriately. Future work will remove some of the limitations of Min-Max LRTA*. For example, our assumption that there is no uncertainty in actuation and sensing is indeed a good approximation for robot navigation tasks in mazes. Future work will investigate how Min-Max LRTA* can be applied to similar tasks in less structured environments in which our assumption is not a good one. Similarly, Min-Max LRTA* works with arbitrary look-aheads. Future work will investigate how it can adapt its look-ahead automatically to optimize the sum of planning and plan-execution time, possibly using methods from (Russell & Wefald 1991).

Experimental Results

Nourbakhsh (Nourbakhsh 1996) has already shown that performing a complete minimax search to solve the goal-directed navigation tasks optimally can be completely infeasible. We take this result for granted and show that Min-Max LRTA* solves the goal-directed 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 better than the upper bound of Theorem 1 suggests. We use a simulation of the robot navigation domain whose interface matches the interface of an actual robot that operates in mazes (Nourbakhsh & Genesereth 1997). Thus, Min-Max LRTA* could be run on that robot. We apply Min-Max LRTA* to goal-directed navigation and localization tasks with two different look-aheads each, namely look-ahead one ($S_{lss} = \{s\}$) and the larger look-ahead from Figure 6. 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. The robot is provided with no knowledge of its start pose and initially senses openings in all four directions. The mazes are constructed so that, on average, more than 1100 poses are consistent with this observation and have thus to be considered as potential start poses. We let Min-Max LRTA* solve the same task repeatedly with the same start pose until its behavior converges.[2] To save memory, Min-Max LRTA* generates the initial u-values only on demand and never stores u-values that are identical to their initial values. Line 5 breaks ties between actions systematically according to a pre-determined ordering on $A(s)$ for all states s. Figure 8 shows that Min-Max LRTA* indeed produces good plans in large domains quickly, 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. Min-Max LRTA* also converges quickly. Consider, for example, the first case, where the initial plan-execution time is worst. In this case, Min-Max LRTA* with look-

[2] Min-Max LRTA* did not know that the start pose remained the same. Otherwise, it could have used 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.

after ...	measuring ...	using ...	Min-Max LRTA* with look-ahead one		Min-Max LRTA* with larger look-ahead (using the method in Figure 6)	
			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 8: Experimental Results

ahead one more than halves its plan-execution time in less than 20 runs. This demonstrates that this aspect of Min-Max LRTA* is important if the heuristic knowledge does not guide planning sufficiently well.

Conclusions

We studied goal-directed navigation (and localization) tasks in mazes, where the robot knows the maze but does not know its initial pose. These problems can be modeled as nondeterministic planning tasks in large state spaces. We described Min-Max LRTA*, a real-time heuristic search method for nondeterministic domains and showed that it can solve these tasks efficiently. Min-Max LRTA* interleaves planning and plan execution, which allows the robot to gather information early that can be used to reduce the amount of planning done for unencountered situations. Min-Max LRTA* allows for fine-grained control over how much planning to do between plan executions and uses heuristic knowledge to guide planning. It amortizes learning over several search episodes, which allows it to find suboptimal plans fast and then improve its plan-execution time as it solves similar planning tasks, until its plan-execution time is at least worst-case optimal. This is important since no method that executes actions before it knows their complete consequences can guarantee a good plan-execution time right away, and 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.

Acknowledgements

Thanks to Matthias Heger, Richard Korf, Michael Littman, Tom Mitchell, and Illah Nourbakhsh for helpful discussions. Thanks also to Richard Korf and Michael Littman for their extensive comments and to Joseph Pemberton for making his maze generation program available to us. This research was sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. 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 or the U.S. government.

References

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

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

Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the National Conference on Artificial Intelligence*, 183–188.

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

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

Good, I. 1971. Twenty-seven principles of rationality. In Godambe, V., and Sprott, D., eds., *Foundations of Statistical Inference*. Holt, Rinehart, Winston.

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

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

Koenig, S. 1997. *Goal-Directed Acting with Incomplete Information*. Ph.D. Dissertation, School of Computer Sci-

ence, Carnegie Mellon University, Pittsburgh (Pennsylvania).

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

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

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

Littman, M. 1996. *Algorithms for Sequential Decision Making*. Ph.D. Dissertation, Department of Computer Science, Brown University, Providence (Rhode Island). Available as Technical Report CS-96-09.

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

Nourbakhsh, I., and Genesereth, M. 1997. Teaching AI with robots. In Kortenkamp, D.; Bonasso, R.; and Murphy, R., eds., *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*. MIT Press.

Nourbakhsh, I. 1996. *Interleaving Planning and Execution*. Ph.D. Dissertation, Department of Computer Science, Stanford University, Stanford (California).

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

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

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

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