Metareasoning in Real-time Heuristic Search

BY

Dylan O'Ceallaigh

B.S., University of New Hampshire (2013)

THESIS

Submitted to the University of New Hampshire
in partial fulfillment of
the requirements for the degree of

Master of Science

in

Computer Science

December 2014

This thesis has been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science by:

Thesis director, Wheeler Ruml, Associate Professor of Computer Science

Radim Bartǒs, Associate Professor of Computer Science

R. Daniel Bergereon, Professor of Computer Science

On Tuesday, December 9th, 2014

Dedication

To my family and friends who encouraged me from a young age. In my ignorant youth I took their words as praise, but as I've grown over the years I've come to recognize them as a challenge to always do my best.

Acknowledgments

Within the confines of this tome rests not just the culmination of my life as a graduate student, but the efforts of all who have supported me in this endeavor. To them I offer my humble thanks, and a recognition far more brief than any of them deserves.

To my advisor, Wheeler Ruml, who has offered me far more assistance than his job description entails, and who has always shown enthusiasm, interest, and willingness to lend an ear. Since I first met him as an undergraduate, I have known that he cares about his students.

To the remainder of the UNH CS faculty who have taught me both the material of their expertise and the value of building expertise of my own.

Finally, to my family, who have supported me since birth, and who never questioned the long and unusual hours for which I was busy during my entire graduate career.

Table of Contents

List of Tables

List of Figures

ABSTRACT

Metareasoning in Real-time Heuristic Search

by

Dylan O'Ceallaigh
University of New Hampshire, December, 2014

In real-time heuristic search, a search agent must simultaneously find and execute a solution to a search problem with a strict time bound on the duration between action emissions and thus the duration of search between said emissions. Because these searches are required to emit actions before their desirability can be guaranteed, resulting solutions are typically sub-optimal in regards to total solution cost, but often superior in terms of the time from start of search to arrival at the goal. Metareasoning is the process of deliberating about the search process, including where to focus it and when to do it. Many modern real-time searches optimistically (and somewhat naïvely) direct the search agent to some state for which the estimated total solution cost appears low at the end of the most recent iteration of search. By employing metareasoning practices, we can hope to replace this optimism with an appropriate amount of skepticism, embracing the already sub-optimal nature of real-time search in such a way that known or expected sub-optimal actions are taken in an effort to direct the agent more effectively in the future. In this thesis we pursue this research direction by presenting and analyzing previous search algorithms and then proposing several metareasoning approaches which show promise in the form of both empirical results and rationalized intuition for minimizing the expected time until arrival at the goal. We evaluate the performance of these approaches thoroughly in order to highlight their strengths and their failings. Finally, we present a number of issues in metareasoning in real-time search which came to light in researching this topic, with the intent of guiding future research along promising avenues.

Chapter 1

Introduction

1.1 Search

Search is a popular problem in computer science in which a search agent must find a path, expressed as a sequence of actions, which will take it from its initial state to some goal state. A search problem is typically expressed using an implicit graph for which the vertices represent states that the agent may occupy and the edges/arcs represent the actions that the agent may execute in order to traverse between connected states. Formally, a standard search problem can be defined as the tuple (S,A,s_0,S_g) where S is a vertex set containing all states in which an agent can reside, A is a set of edges containing all actions adjoining connected states, s_0 is the initial state of the agent, and S_g is the set of possible goal states. Many problems, such as those considered in this thesis, possess only a single goal state. For such problems, S_g is simply a singleton set. Typically, each action has a corresponding cost. These costs are understood to be universally 1 in the case of unit-cost domains, or otherwise variable according to some property of non-unit-cost domains denoted by the cost function c. The sum of action costs along the cheapest path from the current state of the agent s_{agent} to some state s is referred to as the known cost-so-far of s and is denoted as $g(s)$. All of these terms have been made available for future reference in Table 1.1, and a simple example of a search problem is shown in Figure 1-1.

While something as simple as a random walk or depth-first search may eventually

Term	Description
state	A node/vertex of the implicit graph on which search is taking place.
S	The set of all states
$s_0 \in S$	The start state of the agent
$s_{agent} \in S$	The current state of the agent
$S_g \subseteq S$	The set of eligible goal states
action	An edge between two connected states
$A : \{(u,v) \| (u \in S, v \in S)\}$	The set of all actions expressed as tuples of connected states
$c : A \to \mathbb{R}$	The edge cost function
$g : S \to \mathbb{R}$	The known cost-so-far function

Table 1.1: A reference for terms pertinent to graph search

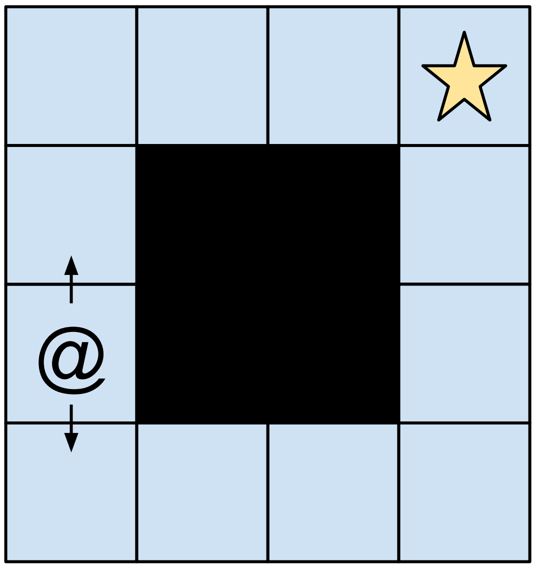


Figure 1-1: A simple example of search in a 2D grid domain. The @ symbol denotes the agent's current/start location, while the star denotes the goal. The agent can move from one state to an adjacent non-occluded state as exemplified by the arrows.

find a path to the goal for simpler or less antagonistic search problems, in order to improve performance and to achieve certain properties of search such as completeness (the guarantee of finding a solution if one exists given infinite resources) and/or optimality (the guarantee that the found solution will be the best possible solution according to a given metric such as total solution cost), many searches follow a common architecture called best-first search (Figure 1-2).

In addition to the generic search terms in Table 1.1, Best-first search introduces several additional terms which we now define. $OPEN$, also referred to as the open list or the search frontier, is a priority queue sorted by some metric (in the case of the generic pseudocode in Figure 1-2, the metric used is x) such that the state s for which $x(s)$ is best is always at the front/top. $OPEN$ may be implemented as any sort-able queue, but typically a heap is used. In addition to $OPEN$, which is used to progress search outwards from the agent, $CLOSED$ keeps track of all of the states that have been generated. $CLOSED$ is a set of all states, including those currently in $OPEN$, for which there is currently a known path from s_{agent}. This serves as a means of ensuring that we do not revisit states via some path for which a superior path has already been found. Finally, *parent* is a field of each state s used to track the predecessor through which s was most recently inserted into $OPEN$. This is important as it allows us to reconstruct the path to the goal after it is found. Not mentioned explicitly in Figure 1-2 but still worth mentioning is the notion of the search space, which is effectively the set of all solutions and partial solutions which will be generated before the goal is reached using the selected metric. Each of these terms has been made available for future reference in Table 1.2. A demonstration of Best-first search using the instance introduced in Figure 1-1 is shown in Figure 1-3.

```
function Search( $x : S \rightarrow \mathbb{R}$ )
1.    $s \leftarrow s_0$
2.    $OPEN \leftarrow \emptyset$
3.    $CLOSED \leftarrow \{s\}$
4.    while $s \notin S_g$
5.        for each successor $s'$ of $s$
6.            if $s' \in CLOSED$
7.                if $x(s')$ is improved with $s'$ as a successor of $s$
8.                    $parent_{s'} \leftarrow s$
9.                    Update the position of $s'$ in $OPEN$
10.           else
11.               Insert $s'$ into $OPEN$
12.               $parent_{s'} \leftarrow s$
13.           $CLOSED \leftarrow CLOSED \cup \{s'\}$
14.       if $OPEN = \emptyset$
15.           Return failure
16.       Pop $s$ off the top of $OPEN$
17.   Return the path generated backwards from $s$ to $s_0$ via parent pointers
```

Figure 1-2: Pseudocode for the generic Best-first search algorithm

Term	Description
search space	The set of all solutions and partial solutions found before search terminates
$OPEN \subseteq S$	The open list or search frontier, a prioritized list of states searching outward from the agent
$CLOSED \subseteq S$	The set of all states which have been generated so far
$parent_s \in S$	The state through which s was most recently inserted into $OPEN$

Table 1.2: A reference for terms pertinent to generic Best-first search

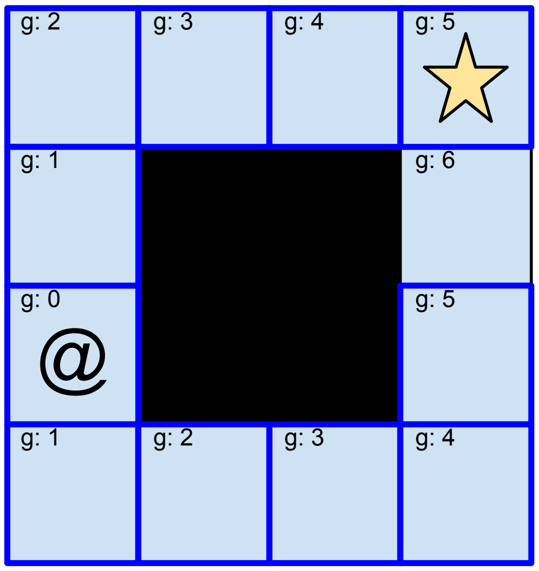


Figure 1-3: A demonstration of Uniform-cost search (Best-first search sorted on low g) in a simple 2D grid instance. The g value of each state is noted in the upper left corner. A bold blue outline denotes that a state is necessarily generated by search before a solution is found.

1.2 Heuristic Search

Some search problems cannot feasibly be solved without slightly more sophisticated means. For these problems, we turn to a subset of search, heuristic search, that uses not only knowledge about the path from s_{agent} to a state s in computing the sorting metric for s, but an estimate about the future path from s to S_g as well. Though there are many metrics by which one might choose to order search, we consider for the time being the problem of optimizing the total solution cost of the found solution. Without the aid of heuristics, we would typically achieve this using something like Uniform-cost search (aka Dijkstra's algorithm), which is equivalent to Best-first search sorted on lowest g. With the use of heuristic estimates, however, we can improve upon this search by pruning the search space and thus reducing the number of expansions and presumably the computation time. The heuristic equivalent of g is h, also referred to as the cost-to-go estimate or the cost heuristic. Where $g(s)$ denotes the known cost of the cheapest path so far from s_{agent} to s, $h(s)$ denotes an estimate of the cost of the cheapest path from s to S_g, computed using knowledge of the problem domain. By adding these two metrics together, we produce the total solution cost estimate function f. By sorting Best-first search on lowest f, we produce one of the first heuristic searches, A* [1]. Depending on the effectiveness of the cost heuristic employed, A* may prune the search space substantially, allowing it to find solutions orders of magnitude faster than the non-heuristic equivalent Uniform-cost search. Additionally, A* is what is referred to as an admissible search, meaning that if the heuristic itself is also admissible, the returned solution is guaranteed to be optimal. A heuristic is called admissible if it never over-estimates the true cost-to-go h^*. In general, the * notation denotes the actual value of what a heuristic estimate is attempting to approximate. Another heuristic which should be noted for its use later in this thesis is the distance-to-go estimate d. Distance in this context is the number

Term	Description
Search admissibility	The property ensuring that a search will return an optimal solution if given an admissible heuristic
Heuristic admissibility	The property ensuring that a given heuristic will never over-estimate the value it is attempting to predict
$h : S \to \mathbb{R}$	The heuristic cost-to-go estimate function
$f : S \to \mathbb{R}$	The total solution cost estimate function
$d : S \to \mathbb{R}$	The distance-to-go estimate function
x^*	For a given heuristic function x, denotes the true value to which the estimate applies

Table 1.3: A reference for terms pertinent to heuristic search

of actions along a given path, regardless of their respective costs. When added to the known distance-so-far metric and applied to Best-first search, we will receive the shortest solution, which is sometimes desirable and generally faster to compute than the cheapest solution. It is also useful in computing certain other estimates as we will show later in this thesis. Each of these terms has been made available for future reference in Table 1.3. A demonstration of A* search using the instance introduced in Figure 1-1 is shown in Figure 1-4. Notice should be taken to the fact that with the addition of the heuristic, fewer states are visited by search before the solution is found.

1.3 Real-time Search

For some problem domains, additional constraints might be imposed on the agent and/or search controller in order to guarantee certain desired properties of behavior.

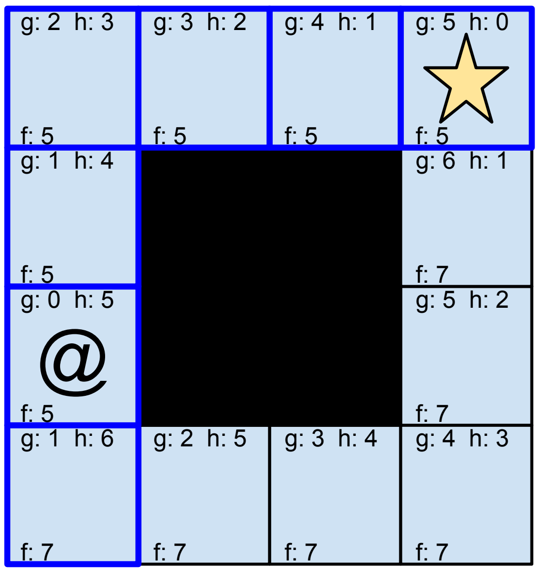


Figure 1-4: A demonstration of A* search in a simple 2D grid instance. The g, h, and f values of each state are noted in the upper left, upper right, and lower left corners respectively. A bold blue outline denotes that a state is visited by search before a solution is found. The heuristic estimate employed is referred to as the Manhattan distance, which is the x offset + y offset of a state relative to the goal state.

The real-time constraint is one such constraint that is popular in the areas of robotics and video game search. The real-time constraint dictates that the agent must emit an action whenever a fixed time bound is reached. Following the same scheme as shown previously, a search problem under the real-time constraint is formally defined as the tuple (S,A,s_0,S_g,t) where t is the per-step time bound for the problem; in other words the length of the time duration between required action emissions. This value is typically set such that an agent selects an action by or before the time when the current action has finished execution. A search algorithm limited by such a constraint must be capable of emitting actions without the guarantee of a complete path to the goal. Assuming that more time is needed to find a guaranteed optimal solution than is allowed by the given time bound, a search algorithm emitting actions under such a constraint cannot guarantee an optimal solution. As such, searches under the real-time constraint should attempt to optimize the expected solution cost (the product of outcome cost and outcome likelihood) rather than the solution cost alone.

1.4 Metareasoning

If we are hoping to optimize the product of outcome quality and outcome likelihood, it is important for our search to be capable of deciding when to gather information and when to commit to a decision. In order to achieve this, we employ the practice of metareasoning. By reasoning about the decision making process, we attempt to better utilize the time available for search. Most decision points have some inherent degree of uncertainty, and it is the goal of the metareasoning process to quantify that uncertainty and evaluate whether it is significant enough to warrant more search, or whether the agent would be best served committing to the seemingly best action available and allocating search to decision points located further down the selected path.

1.5 Goal Achievement Time

While solution cost is still one of the most popular metrics for evaluating the performance of search algorithms, it does not effectively measure the success of real-time searches wherein search duration is also very important. Similarly, a search that finds a solution as quickly as possible is not sufficient if the returned solution takes too long to execute. In order to address this concern, the game time model was proposed [2]. In this model, time is divided into unit action durations, during which the agent may either execute a single action, perform an iteration of search, or do both in parallel. The performance of a search algorithm is measured by the sum of all unit action durations spent searching and all spent executing, minus those during which both were done in parallel. In effect, search is given for free when done in parallel with action execution, under the assumption that the processor would be otherwise idling during action executions. In this model, an offline algorithm such as A* would see a large number of durations spent searching, followed by the shortest possible number of durations spent executing, however there would be no overlap. In a real-time search, however, there would likely be a larger number of durations spent executing, but potentially a better performance overall, given the concurrent nature of the search.

A simplified metric following the same principles has been introduced more recently, dubbed the goal achievement time metric[3]. The performance of a search algorithm according to this metric is said to be simply the amount of time spent between the start of the search and the moment the agent reaches the goal. The evaluation is effectively the same as that in the game time model, but described in simpler terms, and does not require search and execution times to be easily divisible into unit durations. The goal achievement time metric will be the primary method of performance evaluation used in this paper.

1.6 Overview

Throughout the course of this thesis, we examine the history of metareasoning in real-time search and attempt to direct its future. In Chapter 2, we present a number of previously developed heuristic search algorithms of different styles, most involving the real-time constraint and/or metareasoning practices, as well as a comprehensive empirical evaluation and comparison between each with a focus primarily on goal achievement time and total solution cost. We also review in detail the behavior, intuition, and shortcomings of each of these algorithms, followed by a review of what is lacking in more recent algorithms and an introspective on what may be changed or substituted. The goal of this chapter is both to inform the reader of the evolution of real-time search and the place of metareasoning in it and to contextualize the introduction of the new approaches introduced in this paper. In Chapters 3 and 4 we present said novel approaches, each of which uses a meta level decision process in hopes of minimizing goal achievement time. We discuss each of these new algorithms similarly to the review process of the previous algorithms, detailing, in addition to the general structure, behavior, and performance of each, the intuition, assumptions, and possible areas for improvement with the goal of highlighting not only the strengths of the approaches but their failings as motivation for future work as well. In Chapter 5 we present one last series of evaluations for each algorithm using hand-crafted grid instances in order to provide greater insight into not just the performance but also the behavior of each algorithm. Once existing approaches have been thoroughly discussed, Chapter 6 discusses numerous problems in real-time search and the meta level problems it introduces for which the development of concise solutions are deemed important but otherwise outside the scope of this work. Though these problems have not been solved in this work, they have each been considered in detail and should provide novel insight into future research. Finally, in Chapter 7, we conclude by summarizing the contributions of this thesis and highlighting some of the less rigorously investigated ideas from Chapter 6 which seem most promising or important for future research.

Chapter 2

Previous Algorithms

In this chapter we will be discussing a number of previously developed search algorithms and empirically evaluating them across several domains.

2.1 Evaluation

The metrics for evaluation have already been presented and discussed, but the domains and evaluation practices have not. In this section we will present the domains and evaluation practices to be used in upcoming sections.

2.1.1 orz100d

Grid pathfinding is a popular problem in search, prompting Nathan Sturtevant of the University of Denver to compile a large group of existing pathfinding problem instances from popular video games [4]. These instances have been used for evaluation in many papers and are useful because they are known and interesting problems. For this paper, we have taken one such instance, orz100d (Figure 2-1) from the game "Dragon Age: Origins" and evaluated performance across the 25 scenarios (start/goal location pairs) for which the optimal solution cost was greatest. The instance is simple enough that the more powerful algorithms can find solutions quickly and reliably without any one algorithm achieving optimal GAT for all input. The problem is also interesting because it is one of the few observed in which A* has a competitive GAT

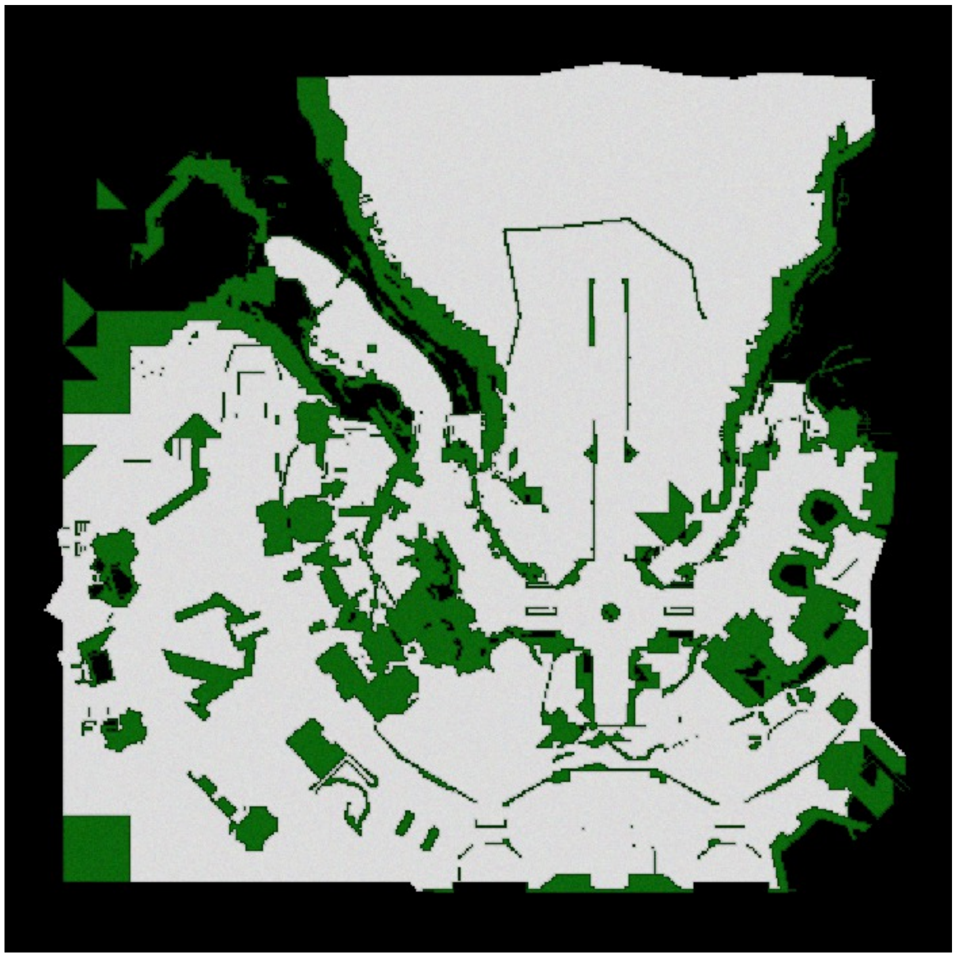


Figure 2-1: Nathan Sturtevant's orz100d grid instance from "Dragon Age: Origins"

for all observed input. Movement for this domain is eight-way, meaning the agent may move from its current state to any of the eight surrounding states unless obstructed.

2.1.2 15-puzzle

Another popular problem in heuristic search is the sliding tile puzzle and specifically, the 15-puzzle variant, the solution for which is shown in Figure 2-2. Instances of the 15-puzzle consist of a 4x4 board with 15 tiles and one empty space, where the tiles are arranged in some order and the goal is to rearrange them in some predefined order by swapping the empty space with an adjacent tile. Not all starting arrangements of

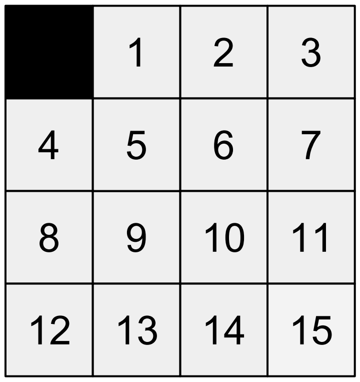


Figure 2-2: The goal state of the 15-puzzle

this problem are solvable, however all of the 100 instances selected for evaluation for this thesis are. Actions are limited to single tile movements.

2.1.3 Platformer

The platformer domain (also called "Mid") was developed by Ethan Burns at the University of New Hampshire. The domain is reminiscent of the 2D platformer genre of video games. The agent is depicted as a small grey knight who can move left and right as well as jump or fall. The agent can control its horizontal movement while airborn, but cannot control how fast it falls, and has a fixed jump height and speed. The goal of the problem is to get the agent to use platforms in order to navigate to a door located somewhere in the level. The problem is particularly interesting because of the dynamic constraints imposed on the agent's movement. Data for evaluation of this domain was gathered from 100 instances, one of which is shown in a screenshot in Figure 2-3.

2.1.4 Traffic

The traffic domain is a grid pathfinding domain designed by Scott Kiesel at the University of New Hampshire in which all obstacles are moving at a constant speed

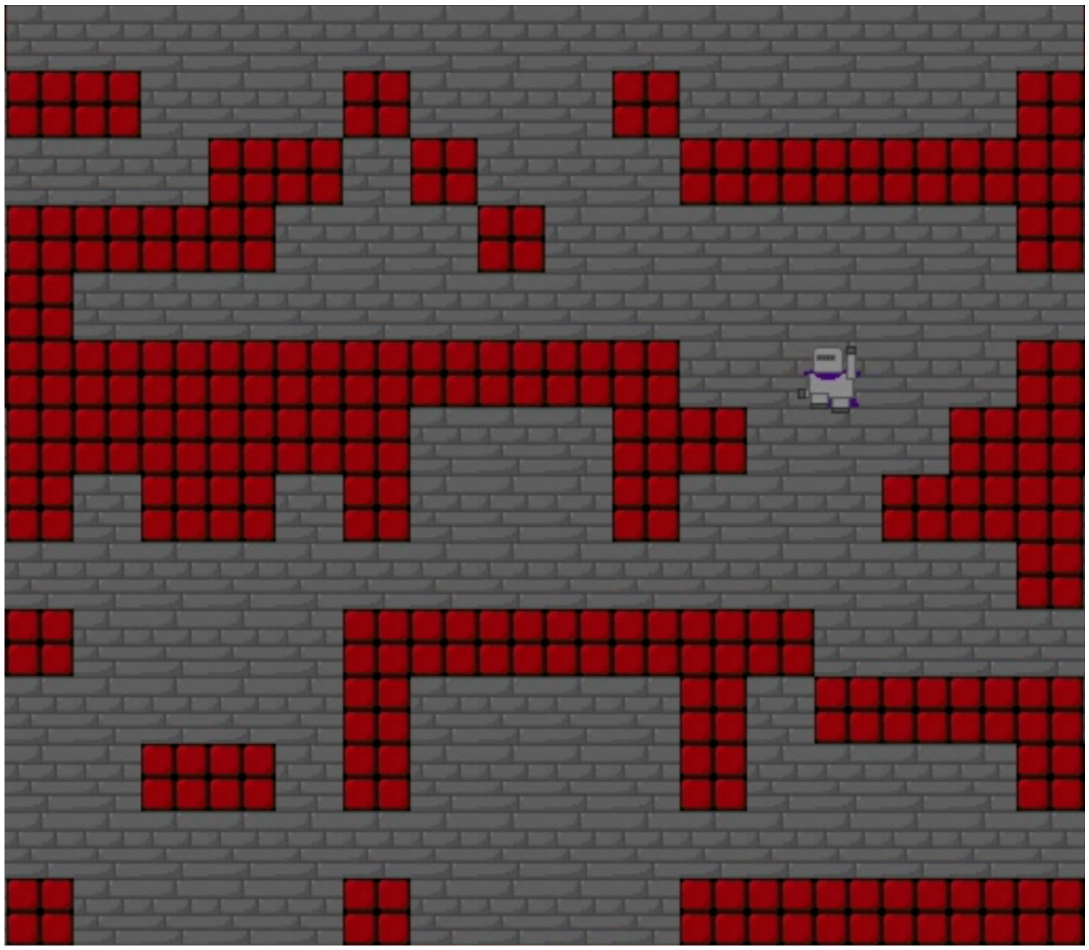


Figure 2-3: A screenshot from one of the test instances for the platformer domain

in a fixed direction. Because time is a dimension of the state space, searches which do not abide by the real-time constraint or otherwise handle dynamic obstacles are given an advantage by effectively choosing when the time progresses and the obstacles move. Obstacles in this domain are "cars" which move along a straight line which loops across map borders. The agent must navigate about these obstacles in order to reach a fixed goal location. Movement is four-way, meaning that the agent can move up, down, left, and right by one tile per action. Obstacle trajectories are deterministic, allowing algorithms to still make use of planning. If a collision occurs between the agent and an obstacle as is occasionally unavoidable, the agent is returned to the start location. Collisions between obstacles are ignored. Evaluation is performed over 25 randomly generated instances. A simplified depiction of what this domain looks like is shown in Figure 2-4.

2.1.5 Practices

For the purpose of empirical evaluation, each algorithm was tested with five trials per instance for each of the aforementioned domains. For each trial, the algorithm was given a distinct number of state expansions per unit duration ranging in powers of 10 from 10^2 to 10^6 inclusive. In effect, the control variable for the different trials is the simulated time taken to execute an action of fixed cost.

The performance of each algorithm was evaluated according to three different metrics: goal achievement time, total solution cost, and failure rate. For goal achievement time and total solution cost, line graphs were created to depict the average performance of each algorithm over all instances of a given domain with respect to the control variable (expansions per unit duration). These average lines are accompanied by error bars denoting the 95% confidence interval relative to the variance in the algorithm's performance across all successful instances. Failure rate is similarly

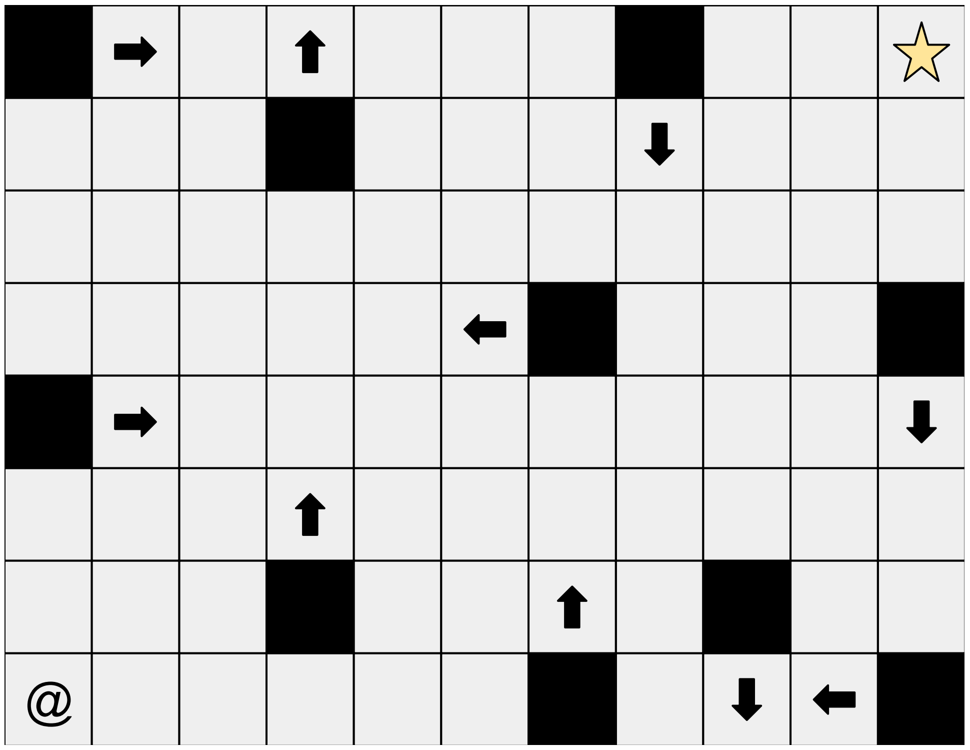


Figure 2-4: A general depiction of the traffic domain, where the agent's start location is denoted by the @ symbol, and the goal location is marked by the star

depicted as a line graph, however rather than average performance with error, the plots simply depict the percentage of tested instances for which the algorithm failed to find a solution.

For each trial, the tested algorithm was given a limit of 6.5 GB of memory and 10 minutes of computation. A trial was said to be a failure if either of these limits was reached before a path to the goal was found. In addition to increasing the failure rate of an algorithm, a failed trial is reflected in the plots in one of two ways. If an algorithm failed all trials for a given instance, that instance would be accounted for in neither the average performance nor the confidence intervals of said algorithm, effectively discounted from the algorithm's performance and reflected only by the failure rate. If the failure was not universal for all tested controls and the offending control was at either extreme, the algorithm's average performance and error were removed from the graph for the relevant control, meaning that the line was effectively cut short at either end. The reason for this is that there is no clear default value to associate with the performance of a failed trial, and so it is best to remove the data when possible rather than obscure the truth with invalid information.

2.2 A*

As previously mentioned, A* was among the first heuristic search algorithms ever developed and despite its age is still widely used as an efficient means of finding the optimal solution for search problems. A* is equivalent to Best-first search (Figure 1-2) ordered on low f, and is detailed as pseudocode in Figure 2-5.

2.2.1 Intuition

For the purpose of discussing the intuition behind A*, we will assume that h is admissible, as one might argue that its performance as a finder of optimal solutions

```
function Search( )
1.   $s \leftarrow s_0$
2.   $OPEN \leftarrow \emptyset$
3.   $CLOSED \leftarrow \{s\}$
4.   while $s \notin S_g$
5.     for each successor $s'$ of $s$
6.       if $s' \in CLOSED$
7.         if $g(s') > g(s) + c(s, s')$
8.           $parent_{s'} \leftarrow s$
9.           $g(s') \leftarrow g(s) + c(s, s')$
10.          Update the position of $s'$ in $OPEN$
11.      else
12.        Insert $s'$ into $OPEN$
13.        $parent_{s'} \leftarrow s$
14.        $g(s') \leftarrow g(s) + c(s, s')$
15.      $CLOSED \leftarrow CLOSED \cup \{s'\}$
16.    if $OPEN = \emptyset$
17.      Return failure
18.    Pop $s$ off the top of $OPEN$
19.  Return the path generated backwards from $s$ to $s_0$ via parent pointers
```

Figure 2-5: Pseudocode for the A* algorithm

is one of the most significant reasons as to why it is still in use today. With this in mind, consider first the previously mentioned Uniform-cost search algorithm which is functionally identical to A* with a heuristic that evaluates to 0 for all states. This particular heuristic is referred to a h_0 and is the weakest non-negative admissible heuristic, meaning that it does the least pruning of any heuristic. The pruning of the search space itself occurs because suboptimal paths will contain states whose f estimates are greater than the value of the goal, and is safe because $f(s)$ is guaranteed to be no greater than $f^*(s)$ for any state s, meaning that if a goal s_g is generated with a path costing $f(s_g)$, all alternative paths for which a goal was not generated must contain some state s such that $f(s) \geq f(s_g)$ and consequently $f^*(s) \geq f^*(s_g)$. It should also be noted that while not explicitly a rule of the algorithm, it is common practice that if two states are added to $OPEN$ with the same f value, higher priority is given to the one with higher g value (and as a result the lower h value). The intuition behind this is that g represents a known value for the cost-so-far whereas h is expected to be lower than the true cost-to-go h^* and so an f value with a greater portion of its value contributed by g is generally expected to be more reliable as a predictor of the true total solution cost metric f^*.

2.2.2 Explanation

Fundamentally, A* is nothing more than Best-first search ordered on f. Lines 1-3 are the setup of the algorithm. Some pseudocode variants will push s onto $OPEN$ at the start, rather than leaving it empty. The reason this is not the case in Figure 2-5 is simply so that the pop can be performed at the end of the loop without causing problems for the first expansion. These two variants are functionally identical. The loop at line 5 is simply a generation of all children of the current best state on the frontier, with lines 6-10 re-placing a state on $OPEN$ if it has been revisited with a better g value, and lines 11-14 ensuring that all newly generated states, regardless of apparent quality, are added to the search frontier. Line 15 similarly ensures that all visited states are accounted for in $CLOSED$. In lines 16 and 17, if we have exhausted the search frontier without reaching a goal, it means that no solution can exist, and so we end search. Otherwise, we progress search by expanding the new best state on $OPEN$ chosen in line 18 until we either fail or reach a goal and return the path to the goal in line 19.

2.2.3 Evaluation

We evaluated the GAT performance, solution cost performance, and failure rate of A*; the results are shown in Figure 2-6, 2-7, and 2-8 respectively. Note that cost and failure rate do not change across action durations. This is because A* is an offline search and does not have any notion of action duration or search iterations. Goal achievement time is affected, however, as an increase in action duration reduces the relative cost of search. If for instance a single action's execution takes as long as the complete offline search, then A* is only forgoing one action by searching all the way to the goal before acting.

2.2.4 Shortcomings

The biggest shortcoming of A*, and all offline search, is that the agent is stuck sitting at the start location for the entire duration of search, even though a reasonable partial path is available long before that. This is of course necessary in order to guarantee an optimal solution, but results in less than desirable goal achievement times for many problems.

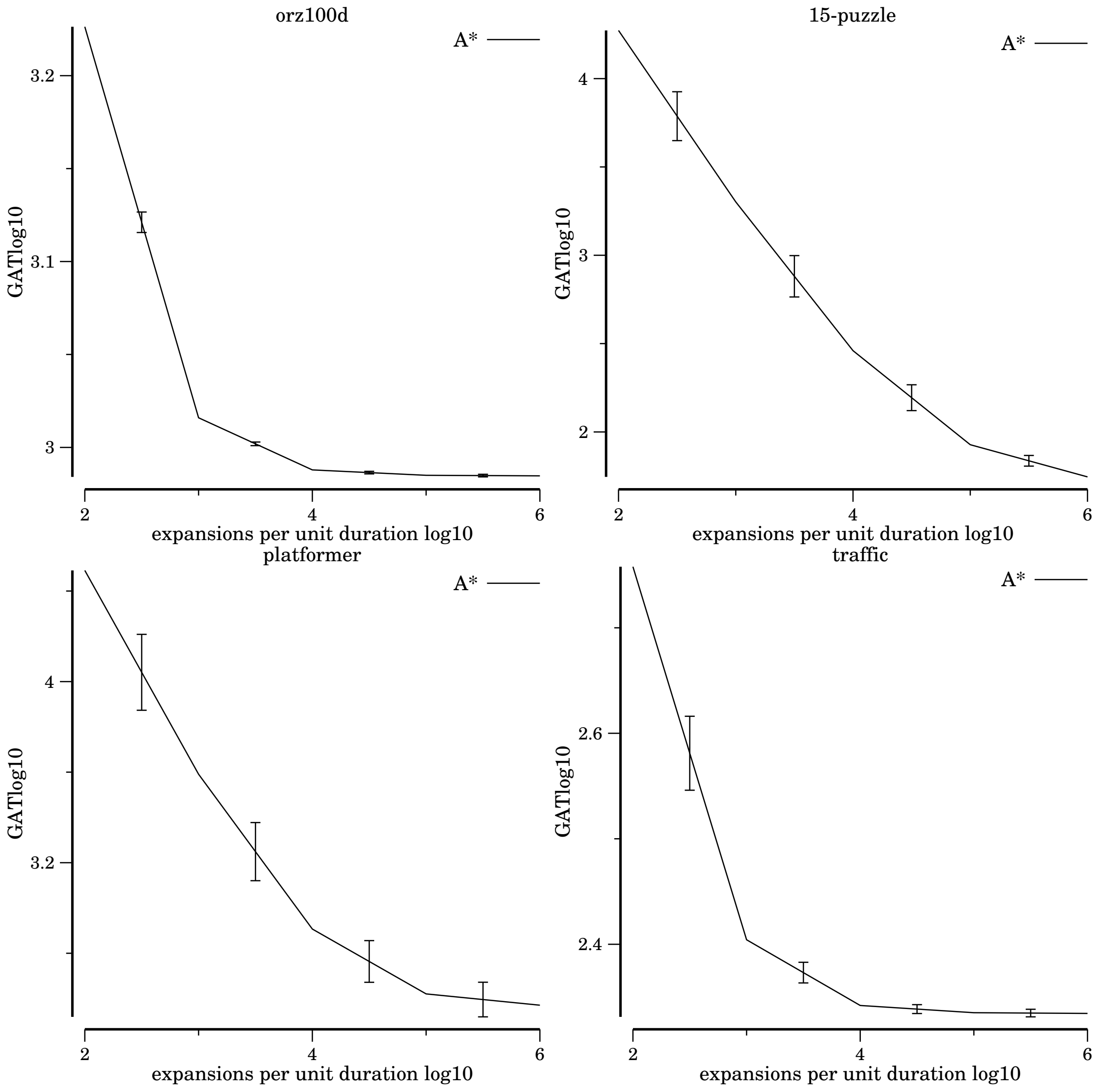


Figure 2-6: GAT performance of the A* algorithm

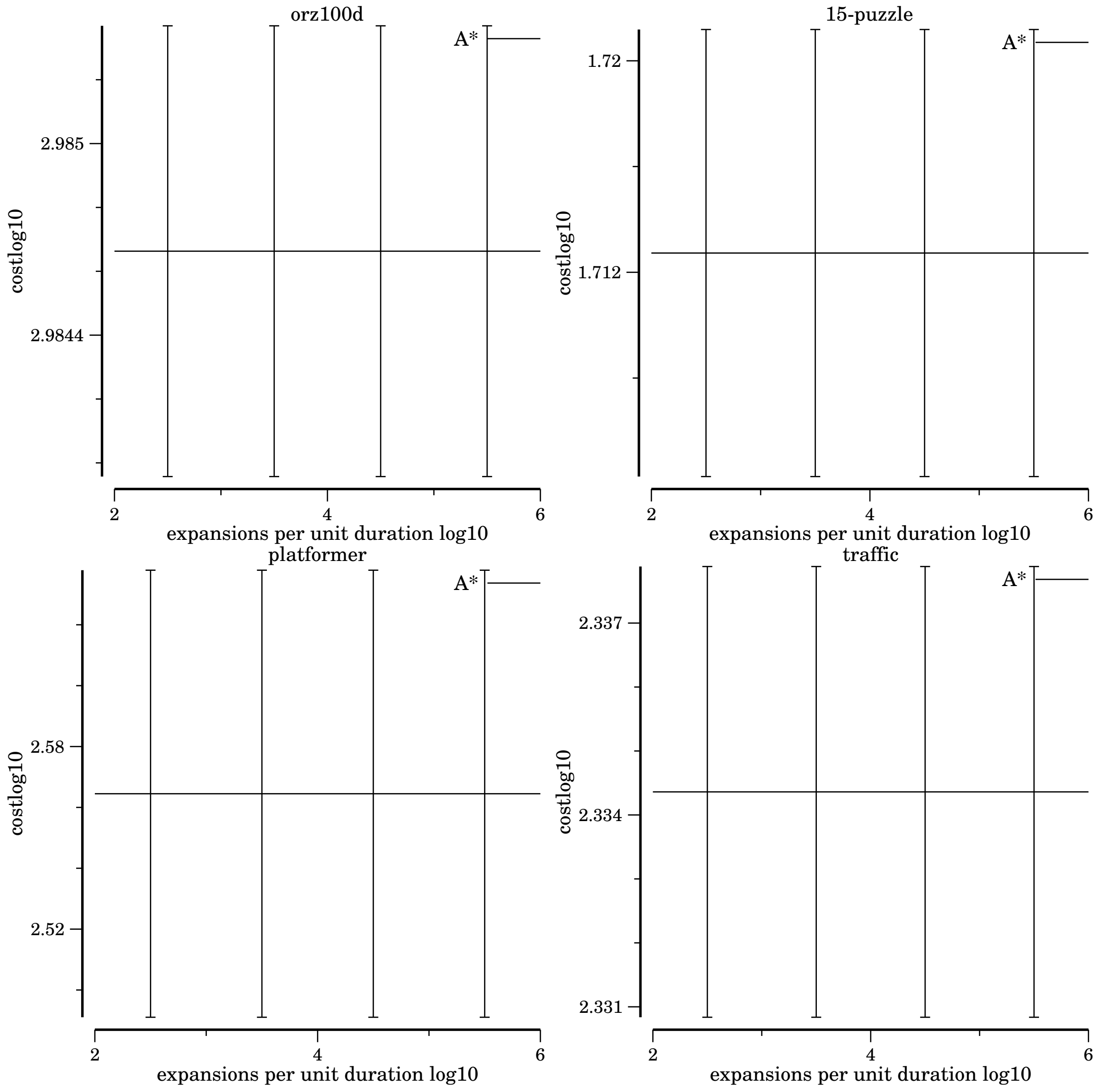


Figure 2-7: Total solution cost performance of the A* algorithm

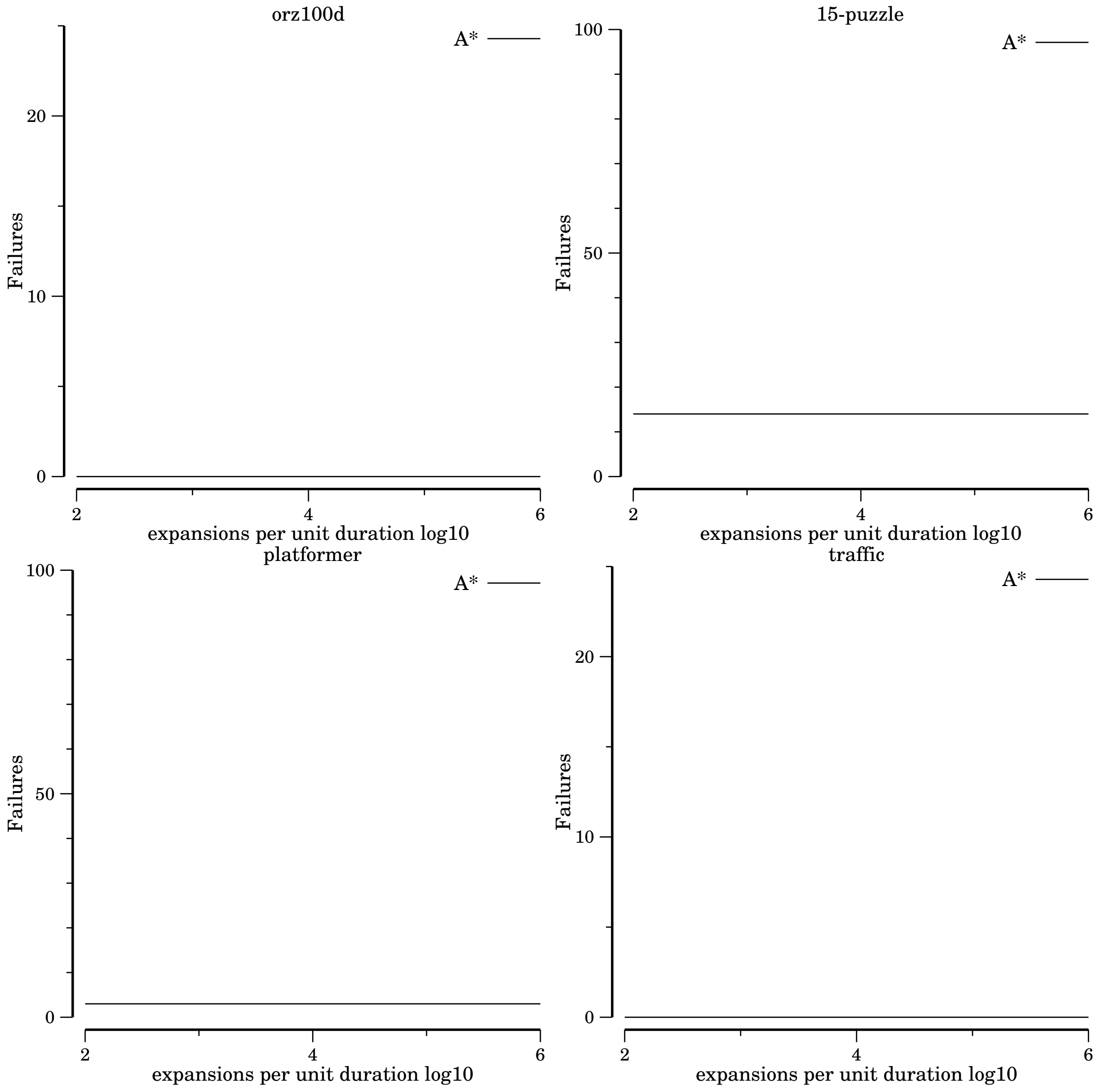


Figure 2-8: Failure rates of the A* algorithm

```
function Search( $depth \in \mathbb{N}$ )
1.  while $s_{agent} \notin S_g$
2.      for each successor $s'$ of $s$
3.          Perform alpha-pruned search from $s'$ toward $S_g$ up to depth $depth$
4.          $f'(s') \leftarrow$ the resulting $f$ value of the best state from alpha-pruned search
5.      If LRTA*, $h(s_{agent}) \leftarrow f'$ value from best $s'$, otherwise $f'$ value from second best $s'$
6.      $s_{agent} \leftarrow$ best $s'$
```

Figure 2-9: Pseudocode for the LRTA* algorithm

2.3 (L)RTA*

Two of the oldest and most well known real-time search algorithms are Real-Time A* (RTA*) and Learning Real-Time A* (LRTA*) [5][6][7]. These algorithms introduced the concept of updating cost estimates during search in order to avoid getting stuck in heuristic local minima while still allowing actions to be taken before a full plan is available. The concept is simple and fairly intuitive, and is outlined in Figure 2-9.

2.3.1 Intuition

The primary contribution of RTA* and LRTA* which allows them to function in real-time is the learning which allows the agent to escape local minima. By inflating the heuristic estimates of states each time they are visited, we ensure that the agent will not continue to favor any one state over its neighbors indefinitely. This is necessary to ensure that we continue to explore until we reach the goal without getting stuck in an infinite loop.

Consider Figure 2-10, which depicts conceptually a search agent (denoted by the

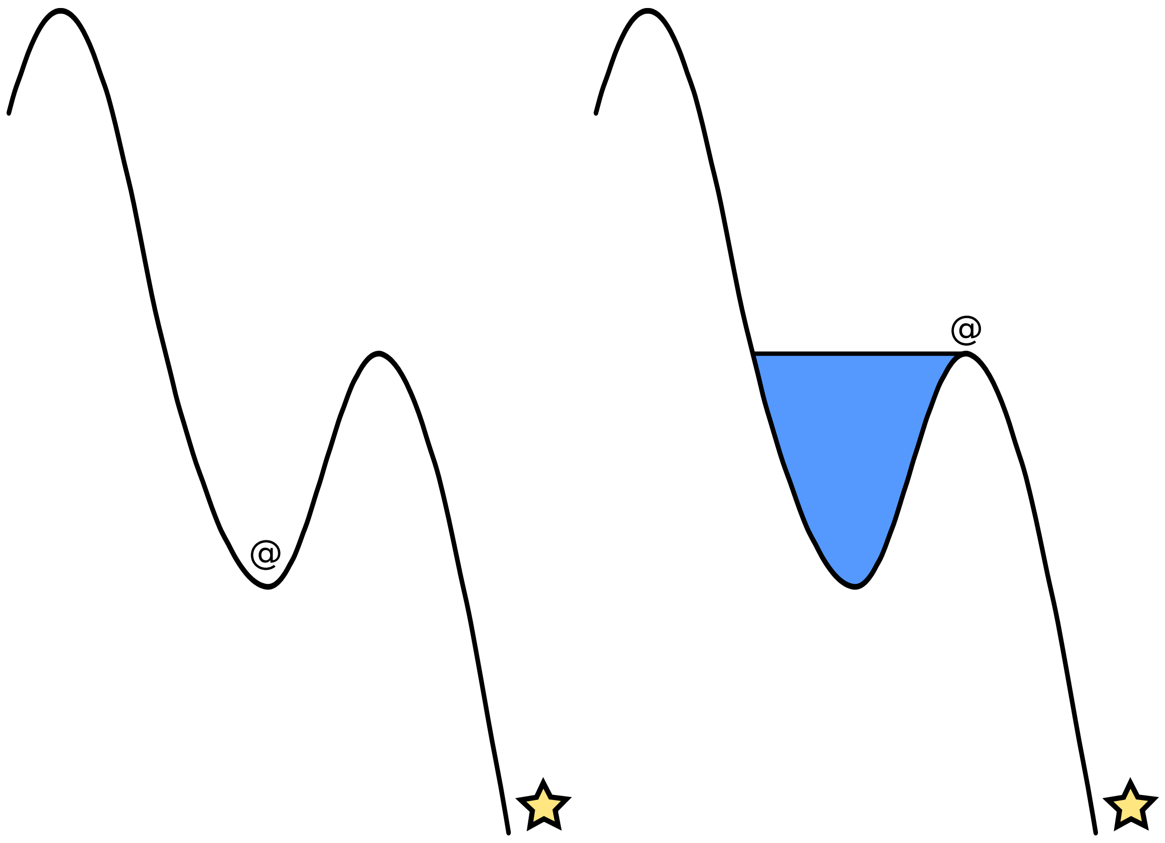


Figure 2-10: (left) An agent (@ symbol) stuck in a local minima caused by a misleading heuristic (right) After filling in the local minima through learning (blue) the agent is able to continue toward the goal (star)

@ symbol) trying to navigate toward the goal (denoted by the star). The agent has no immediate neighbors which appear closer to the goal than its current location, and so it remains stuck. After learning, the local minima is filled, allowing the agent to once again follow the heuristic to the goal. This is the rationale behind the behavior of RTA* and LRTA*, as well as subsequent real-time search algorithms.

2.3.2 Explanation

Like other searches, the body of (L)RTA* begins with a check to see whether or not a goal state has been reached (line 1). If no such state has been reached, we expand the agent's current state (line 2) and perform a depth limited search on each successor up to some user decided depth (line 3). Alternatively, a regular A* search without pruning may be used, but Korf outlines a proof that significant pruning can be done to minimize the amount of lookahead done to find the best state under the given successor up to the specified depth [7]. This approach has an advantage in the context of (L)RTA*'s update process in that it reduces the number of states which must be visited before reaching the best successor at the given depth. At the end of the lookahead process for each successor s' we make a note of the lowest f value at the selected depth and record it as $f'(s')$. Lines 4 and 5 are where the actual learning is done. We look among each successor of s_{agent} and update $h(s_{agent})$ to be either the lowest (LRTA*) or the second lowest (RTA*) of the f' values found in the previous step. By selecting the lowest f', repeated trials are guaranteed (in domains represented by undirected graphs) to converge to the optimal solution [7]. Backing up the second lowest value has no such guarantee, but has been shown empirically to provide better performance on the first iteration, which is generally more practical. Finally in line 6 the agent moves to its best child and repeats search until it reaches a goal state.

2.3.3 Evaluation

Given the slow nature of (L)RTA*'s learning process, it routinely hit the computation limit well before the memory limit. The issue was so severe that, given the same memory limit and unlimited computation time, many trials would take on the order of hours to days to reach either success or failure. The reason for this is a combination of RTA* and LRTA* updating only one cached h value per iteration and the exhibition of a phenomena in real-time search referred to as scrubbing. Scrubbing is used to describe the behavior of an agent guided by real-time search in which it revisits previous states many times over. Every time the agent revisits a previous state in either RTA* or LRTA*, the backed up h value of s_agent is cached as an update rather than an addition, meaning that not only has the agent not made it closer to S_g, but also no new memory is occupied. The resulting performance relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to A* is shown in Figures 2-11, 2-12, and 2-13 respectively.

It should be immediately clear that, despite being given more generous limits to its computation than any other algorithm, both RTA* and LRTA* failed the vast majority of the trials. It is worth noting, however, that even given this generally abysmal performance, the average GAT of the successful trials for the 15-puzzle is still better than with A*. This should effectively illustrate the importance of executing incomplete solutions in minimizing goal achievement time.

2.3.4 Shortcomings

While RTA* and LRTA* effectively accomplish their task of pushing the agent out of local minima, the efficiency of the learning process is somewhat lacking. For larger depths, more work is being done in the lookahead step, providing more accurate heuristic backups, but only the agent's current state benefits, and the effort is oth-

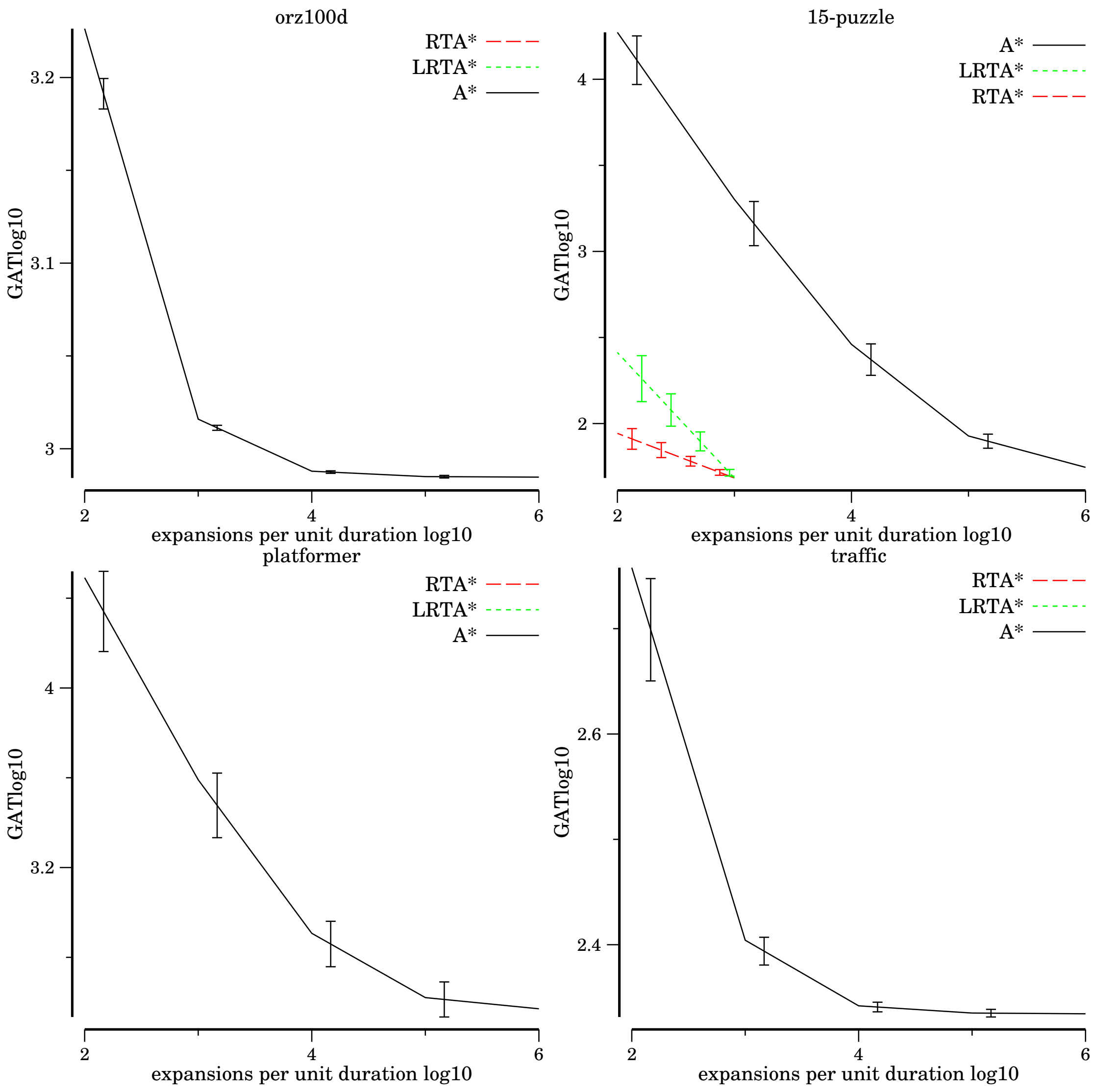


Figure 2-11: GAT performance of the RTA* and LRTA* algorithms

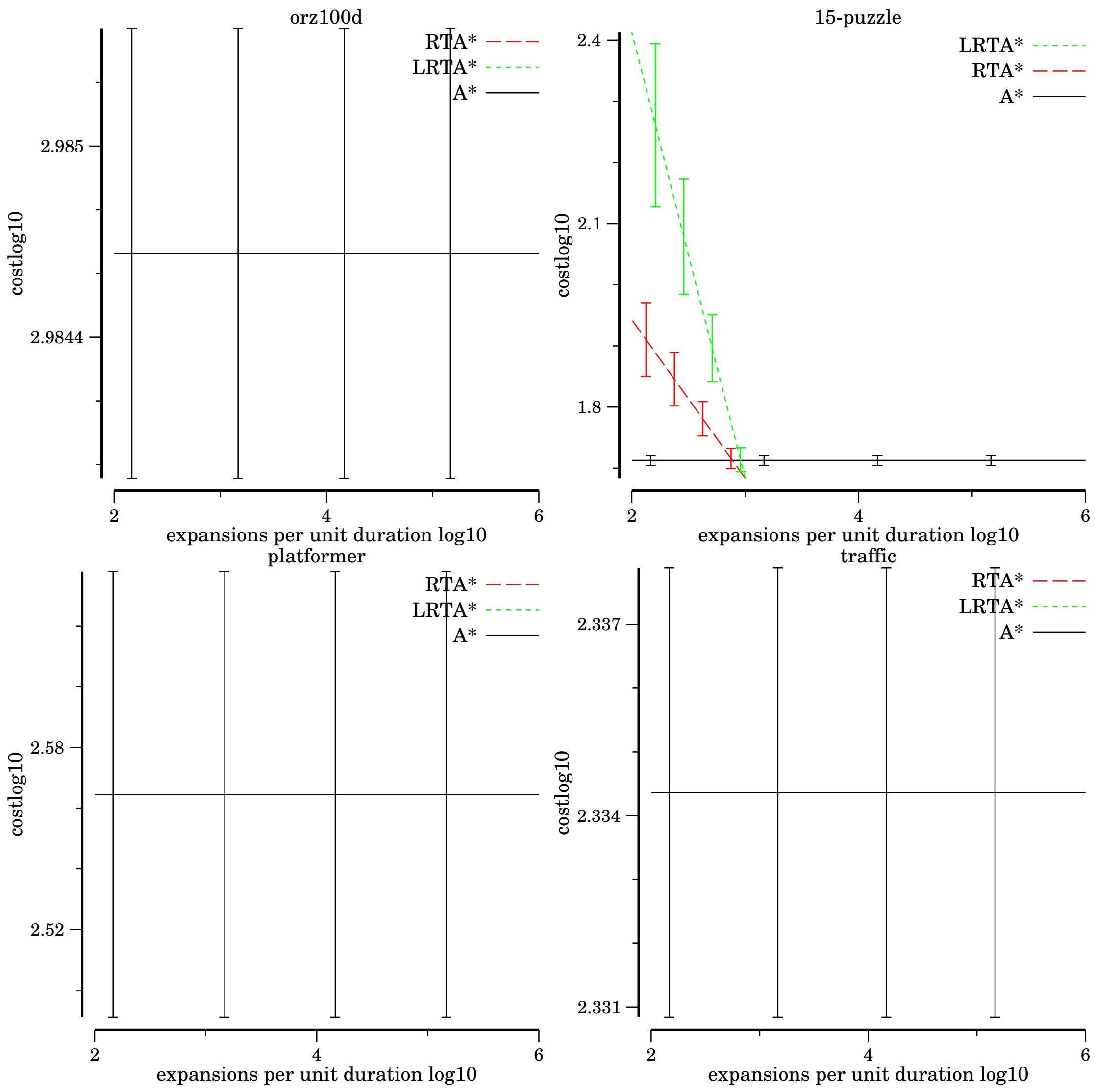


Figure 2-12: Total solution cost performance of the RTA* and LRTA* algorithms

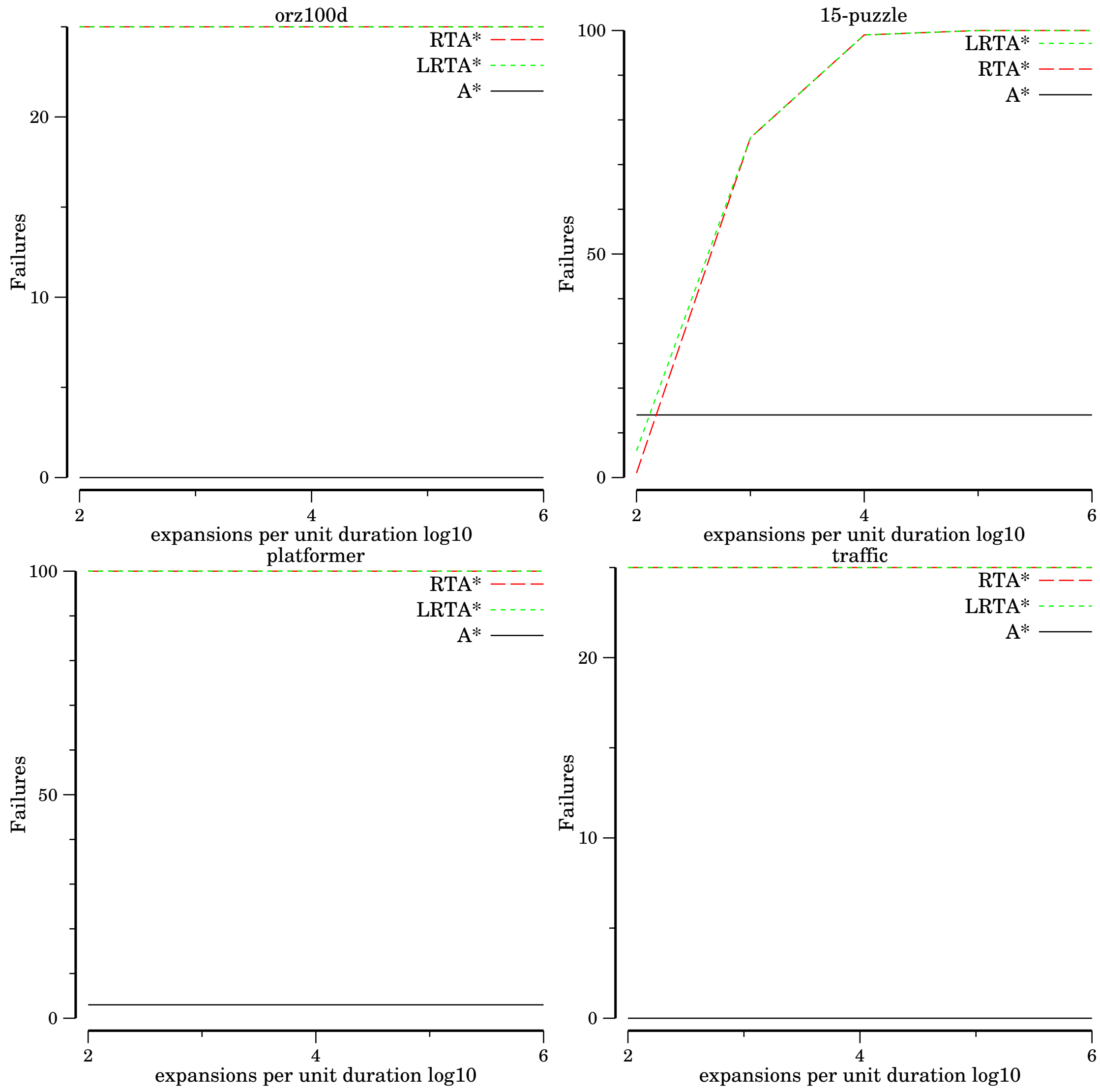


Figure 2-13: Failure rates of the RTA* and LRTA* algorithms

Term	Description
lookahead $\in \mathbb{R}$	The number of expansions remaining before the next action must be emitted
local search space $\subseteq S$	The set of states which have been generated since the start of the current iteration of search

Table 2.1: A reference for terms pertinent to LSS based search

erwise wasted. This also results in revisiting states through search many times over as the lookahead step will continue to expand many of the same states until they are given inflated h values of their own.

2.4 LSS-LRTA*

Many of the more recent developments in real-time search have focused on improving the learning process. One of the most notable and effective modifications came with the introduction of Local Search Space LRTA* (LSS-LRTA*) [8]. Unlike its predecessor, which only updates the cost estimate of the agent's current state for each iteration, LSS-LRTA* uses a Dijkstra-like approach in order to update the h values for all states generated in the agent's entire local search space (Table 2.1) in a single iteration.

2.4.1 Intuition

The improved learning process of LSS-LRTA* results in drastically superior search performance over its predecessor, and is the fundamental underlying structure of learning in many modern real-time searches. The intuition behind this change is simple: because the local search space is bounded by the real-time bound, so is a

```
function Search( )
1.    while $s \notin S_g$
2.        Perform $BFS(f)$ for $lookahead$ expansions
3.        if $OPEN = \emptyset$
4.            stop
5.        $s' \leftarrow$ state in $OPEN$ with the lowest $f$
6.        Run modified Dijkstra's algorithm on $CLOSED$
7.        Continue to move the agent along the path from $s$
          to $s'$ until it reaches $s'$ or action costs change.
8.        $s \leftarrow$ The current state of the agent
```

Figure 2-14: Pseudocode for the LSS-LRTA* algorithm

polynomial time dijkstra backup procedure on that local search space. Additionally, this extra learning will greatly increase the influence of learning on the agent's behavior, as it will push it out of a much larger area of heuristic inflation.

2.4.2 Explanation

The functional behavior of LSS-LRTA* is similar to that of LRTA*. Once again the search continues until a termination condition (line 1) is reached. Most of the novelty of the algorithm is contained in two parts (lines 2 through 6). The first part is the expansion of the local search space through the use of a bounded A* search (line 2). This step should result in some frontier which is standardly referred to as $OPEN$. If such a frontier does not exist, it means that either no states were expanded, or all reachable states were expanded and no goal was found. We perform a check (line 3) to see if such a condition was reached and terminate accordingly (line 4) before proceeding. If no such situation occurs, we begin the second major part of the algorithm, the learning process and shortest path selection. First, we select the most promising state on the frontier as the short term goal s' for the agent (line 5). Next, we iterate through the search space from the frontier inward updating each state's path pointer to its most promising (lowest f) child, and updating heuristic estimates to be equal to the h of their best child plus the cost of traversing to that child (line 6). Finally, we follow the given path to s' for as long as possible (line 7) and prepare for the next iteration of search (line 8).

2.4.3 Evaluation

We evaluated the performance of LSS-LRTA* relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to A* and (L)RTA*; the results are shown in Figures 2-15, 2-16, and 2-17 respectively.

It is worth noting that in Figure 2-15, LSS-LRTA* dominates or at very least is competitive with A* in all domains except for the orz100d grid instance. In orz100d, A* does not take long to find a solution and converges to a single action duration quite quickly. It should also be noted that there is a spike in the GAT and cost of LSS-LRTA* for the traffic domain, as is the case for other real-time searches as well. This is a unique behavior of the traffic domain presumably caused by the restarting process which occurs when the agent collides with an obstacle. Because the domain resets to the start state upon collision, a real-time search with lookaheads just barely insufficient for finding the goal during the first iteration may lead the agent along a long and promising path which ultimately ends in an unavoidable collision, forcing the agent to then restart from the initial state. The real-time searches presumably avoid this trap for other action duration lengths as the antagonistic area is only on

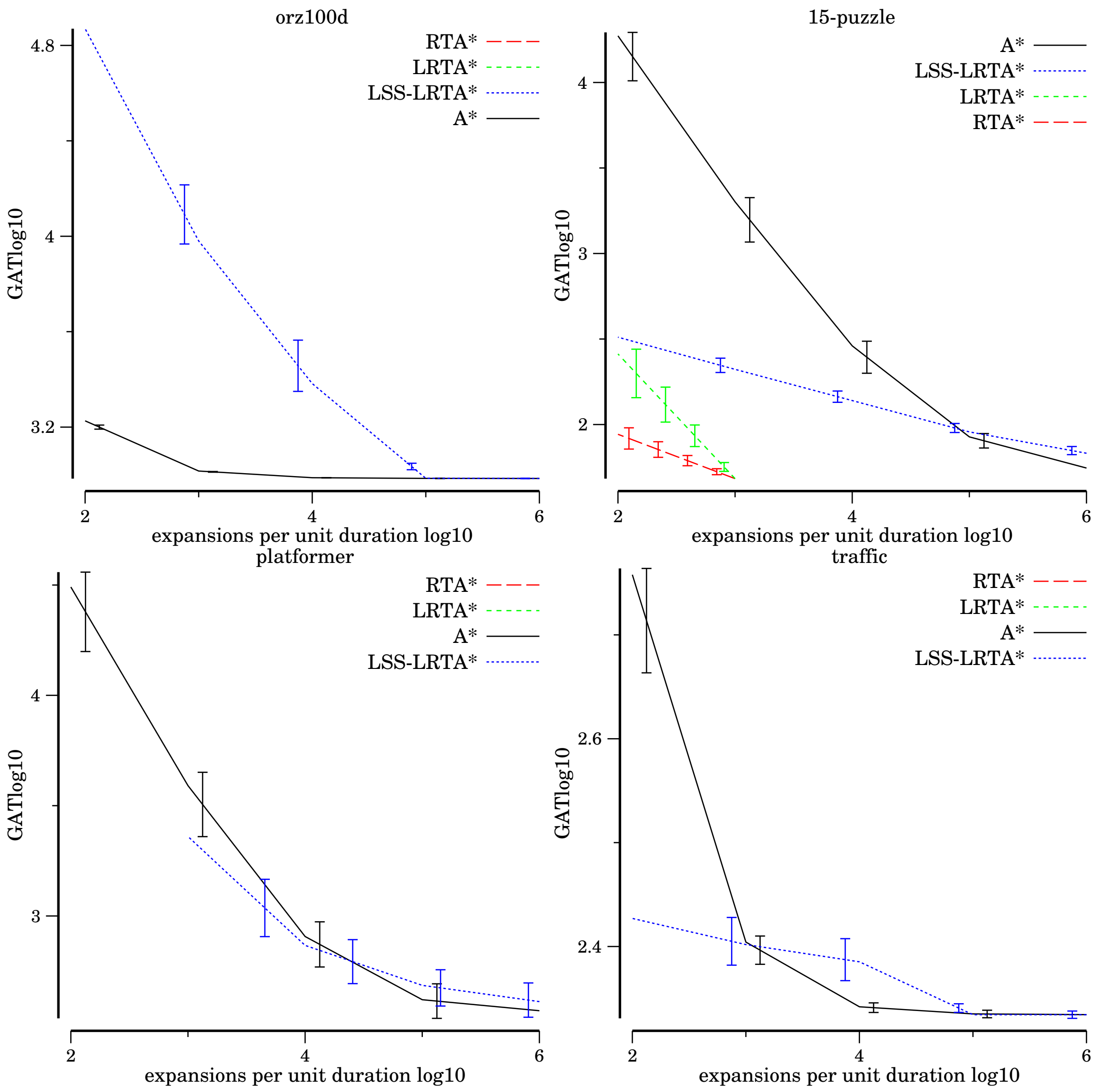


Figure 2-15: GAT performance of the LSS-LRTA* algorithm

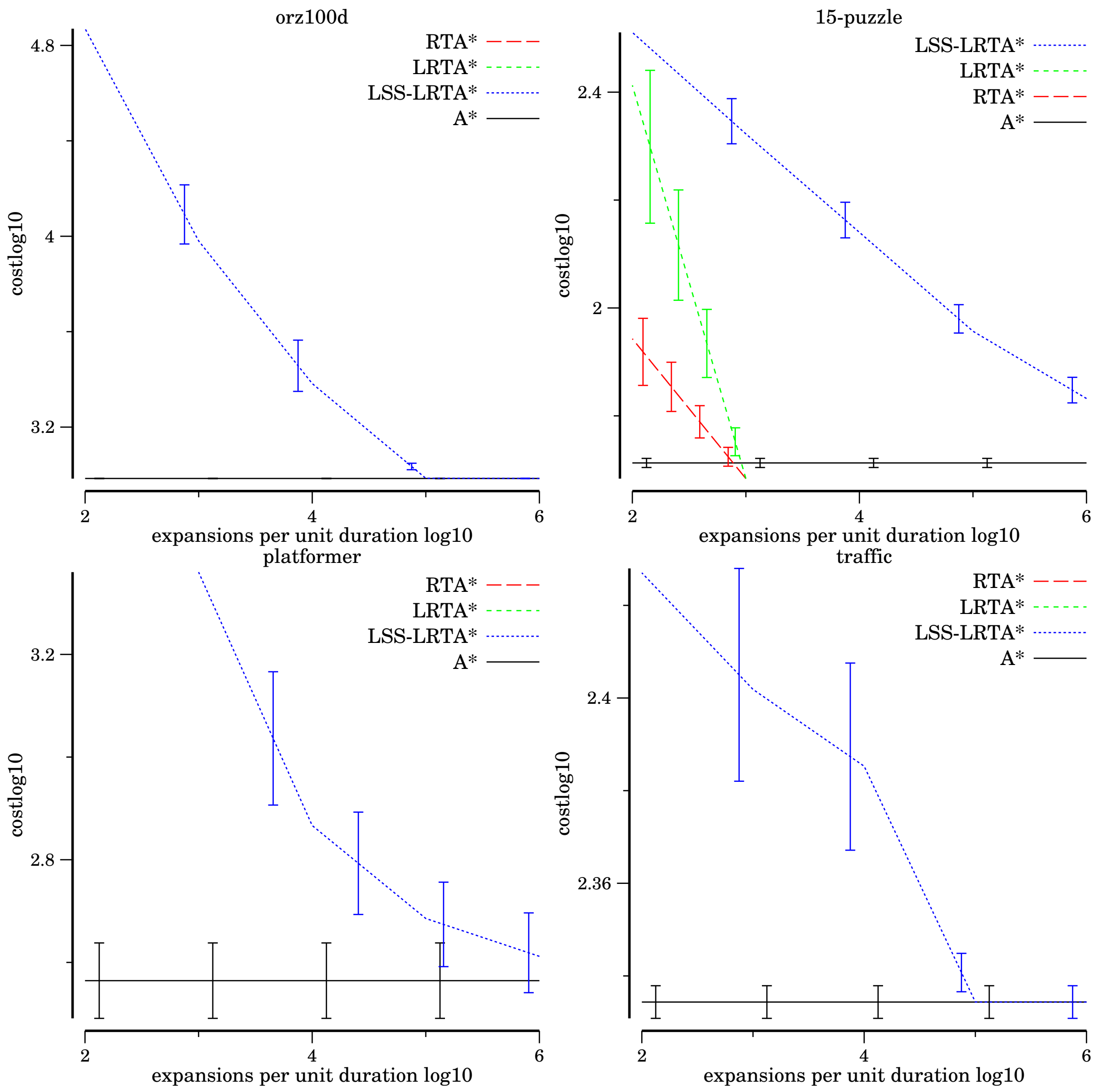


Figure 2-16: Total solution cost performance of the LSS-LRTA* algorithm

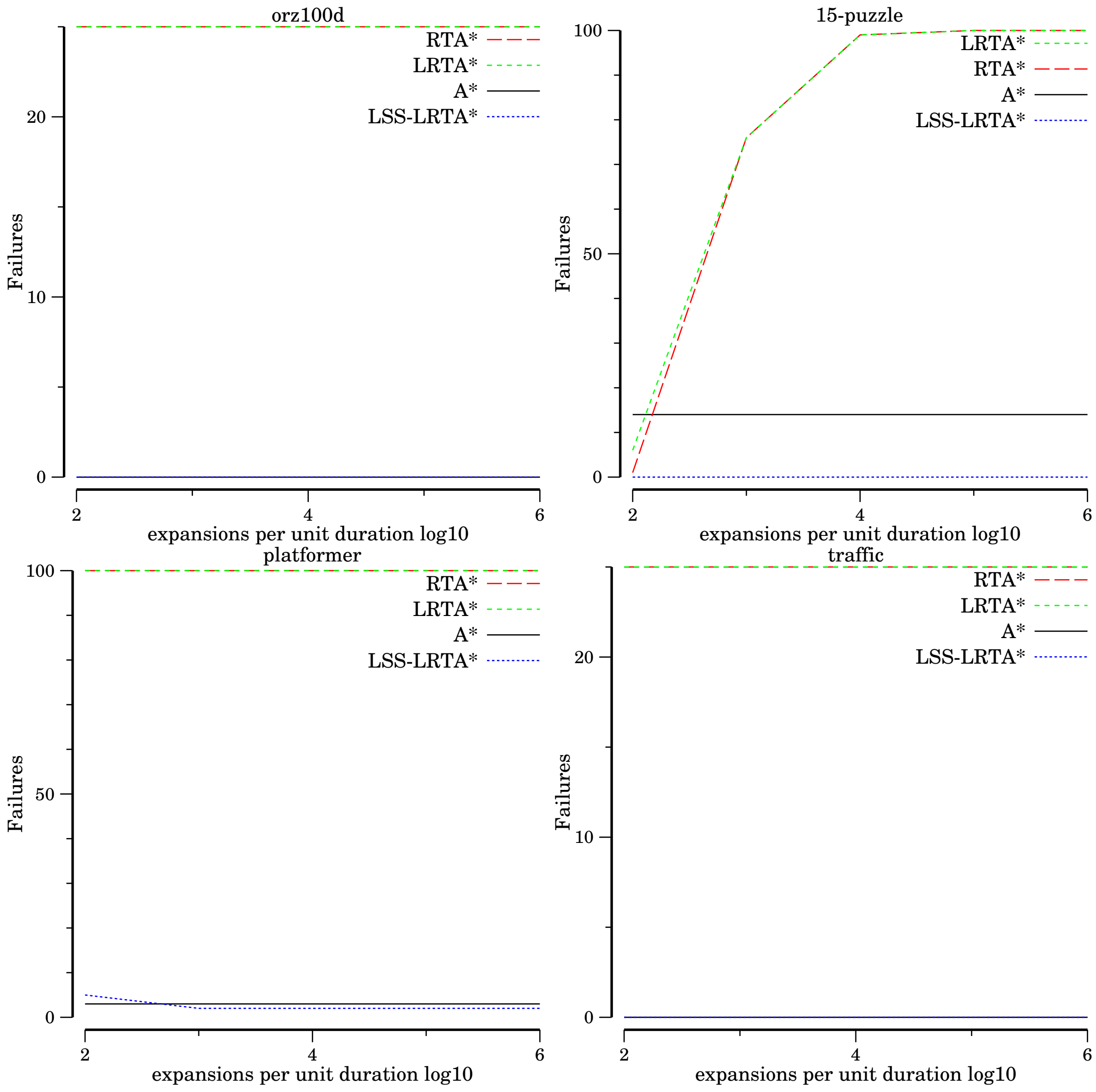


Figure 2-17: Failure rates of the LSS-LRTA* algorithm

the search frontier for a short while, and not problematic unless it is generated at the end of an iteration.

2.4.4 Shortcomings

LSS-LRTA*, though vastly superior to RTA* and LRTA* in its learning process, is still susceptible to misdirection by an under informed heuristic. In fact, because LSS-LRTA* by default commits to a path all the way to the frontier rather than only a single step, the agent may move a substantial distance in the wrong direction before additional search directs it back the way it came. This not only causes the agent to exhibit behavior which looks strange and irrational to an outside observer (which is undesirable for many problems such as video game path planning), but can more unique states to be generated as well, exhausting more memory than A* (hence the higher failure rates for bigger unit durations).

2.5 Dynamic $\hat{f}$

2.5.1 Intuition

Recently, promising results have been shown using online heuristic correction with an algorithm called Dynamic $\hat{f}$ (Burns et al., Manuscript under review). The intuition behind Dynamic $\hat{f}$ is simple enough: an admissible heuristic is necessary for finding the optimal cost solution, but because we are already committed to suboptimality, we should instead focus on building as accurate a heuristic as possible, regardless of admissibility.

2.5.2 Explanation

Dynamic $\hat{f}$ is adapted closely from LSS-LRTA*, with two key changes. The first modification is in line 2 of Figure2-18, the introduction of the $\hat{f}$ metric. In addition

```
function Search( )
1.    while $s \notin S_g$
2.        Perform $BFS(\hat{f})$ for $lookahead$
          expansions on $OPEN$
3.        if $OPEN = \emptyset$
4.            stop
5.        if $P = \emptyset$
6.            $s' \leftarrow$ state in $OPEN$ with the lowest $\hat{f}$
7.            Run modified Dijkstra's algorithm on $CLOSED$
8.            $P \leftarrow$ path from $s$ to $s'$
9.        Pop and emit the head of $P$
10.       if edge costs change
11.           $P \leftarrow \emptyset$
12.           $OPEN \leftarrow \{s\}$
```

Figure 2-18: Pseudocode for Dynamic $\hat{f}$

to replacing f as the method for sorting $OPEN$, there is an implicit procedure within the call to BFS on line 2 wherein the heuristic h must be corrected to compute $\hat{h}$. The second change is in the interleaving of search and execution, with the introduction of the dynamic lookahead paradigm. Like the traditional LSS-LRTA* style static lookahead, Dynamic-$\hat{f}$'s dynamic lookahead involves queueing a path to some state on the search frontier (lines 6-8) and moving the agent along that path by executing the queued actions. However, when using a static lookahead, a constant search duration is given, despite the fact that the time taken to execute the queued actions is dependent on the queued path, which is not constant. The dynamic lookahead paradigm rectifies this by searching until the entire path has been traversed, or the remaining queued actions are made infeasible in some way (lines 9-12).

2.5.3 Evaluation

We evaluated the performance of Dynamic-$\hat{f}$ relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to previous algorithms; the results are shown in Figures 2-19, 2-20, and 2-21 respectively. Although the algorithm was introduced as a real-time $\hat{f}$ directed search with dynamic lookaheads, these two features are not interdependent, and the dynamic lookahead paradigm can be applied to any LSS based real-time search. For these reasons, future evaluations include both the static lookahead and dynamic lookahead variants of each applicable algorithm. For the purposes of avoiding clutter in the plots, RTA* and LRTA*, which have been previously shown to be dominated nearly universally by LSS-LRTA*, have been removed from future plots.

In general, Dynamic-$\hat{f}$ returns solutions with quality similar to LSS-LRTA*, with the exception of the 15-puzzle, where it dominates for shorter action durations. It is worth noting, however, that in Figure 2-21, Dynamic-$\hat{f}$ is significantly less prone to

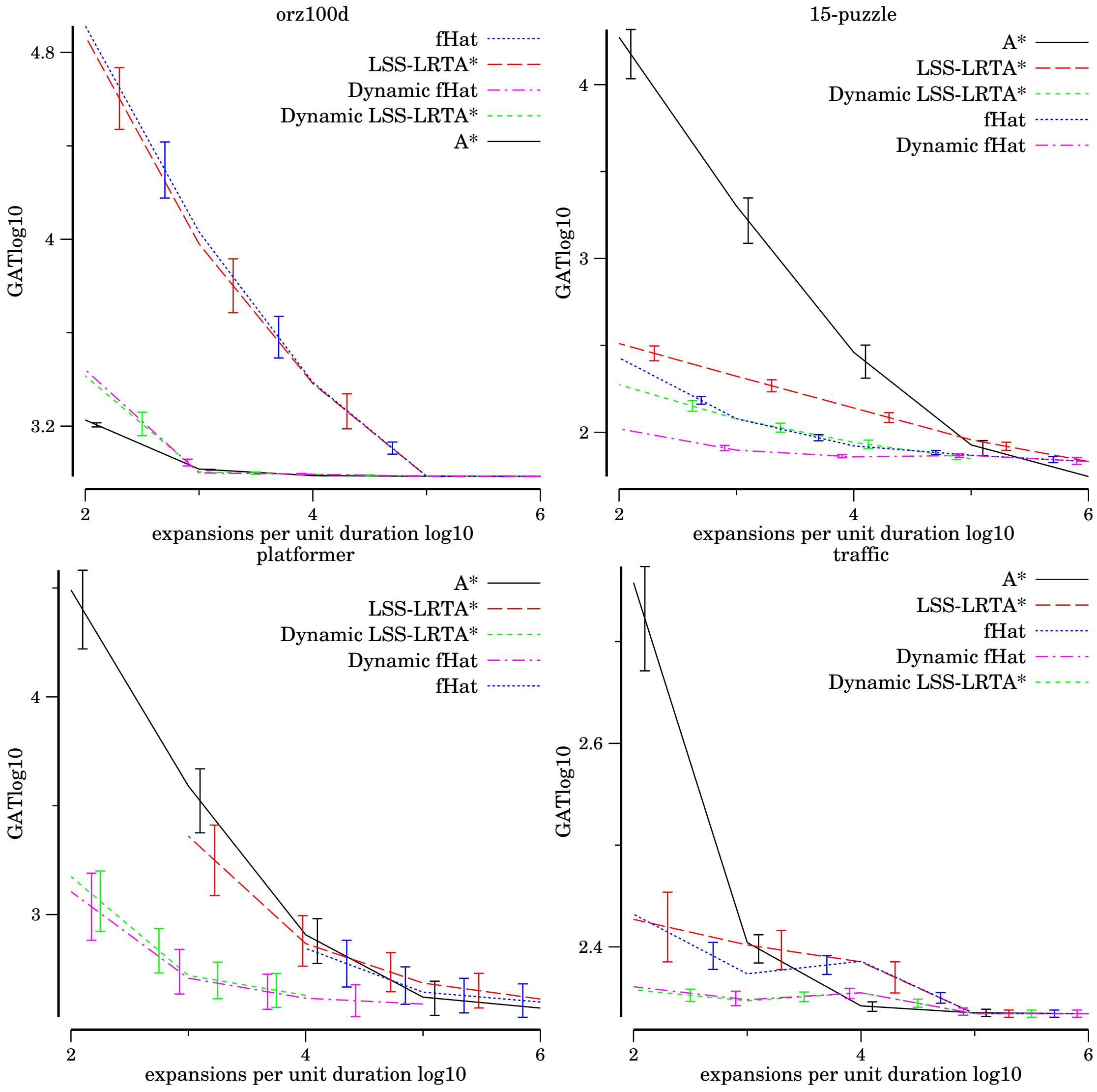


Figure 2-19: GAT performance of the Dynamic-$\hat{f}$ algorithm

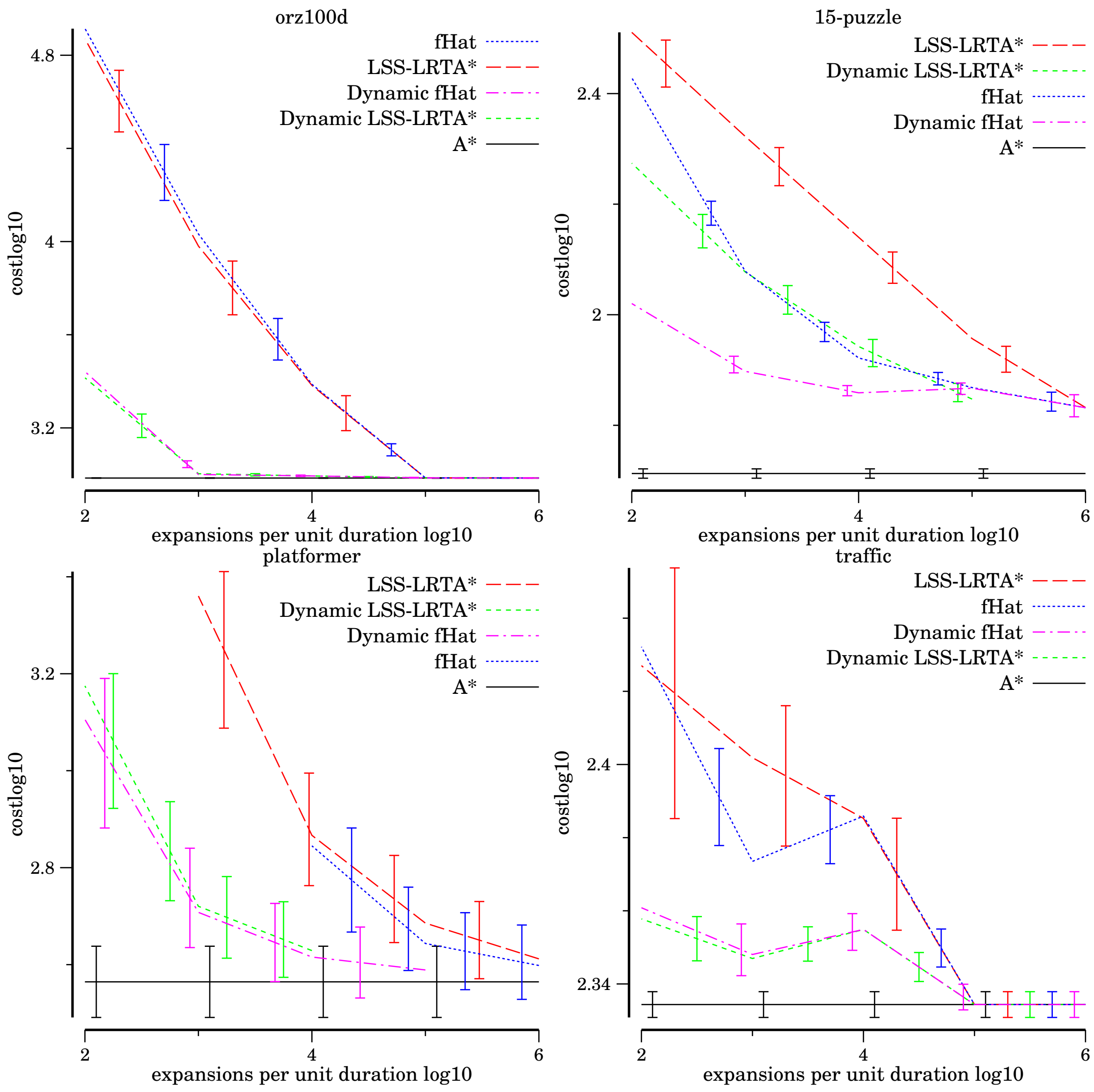


Figure 2-20: Total solution cost performance of the Dynamic-$\hat{f}$ algorithm

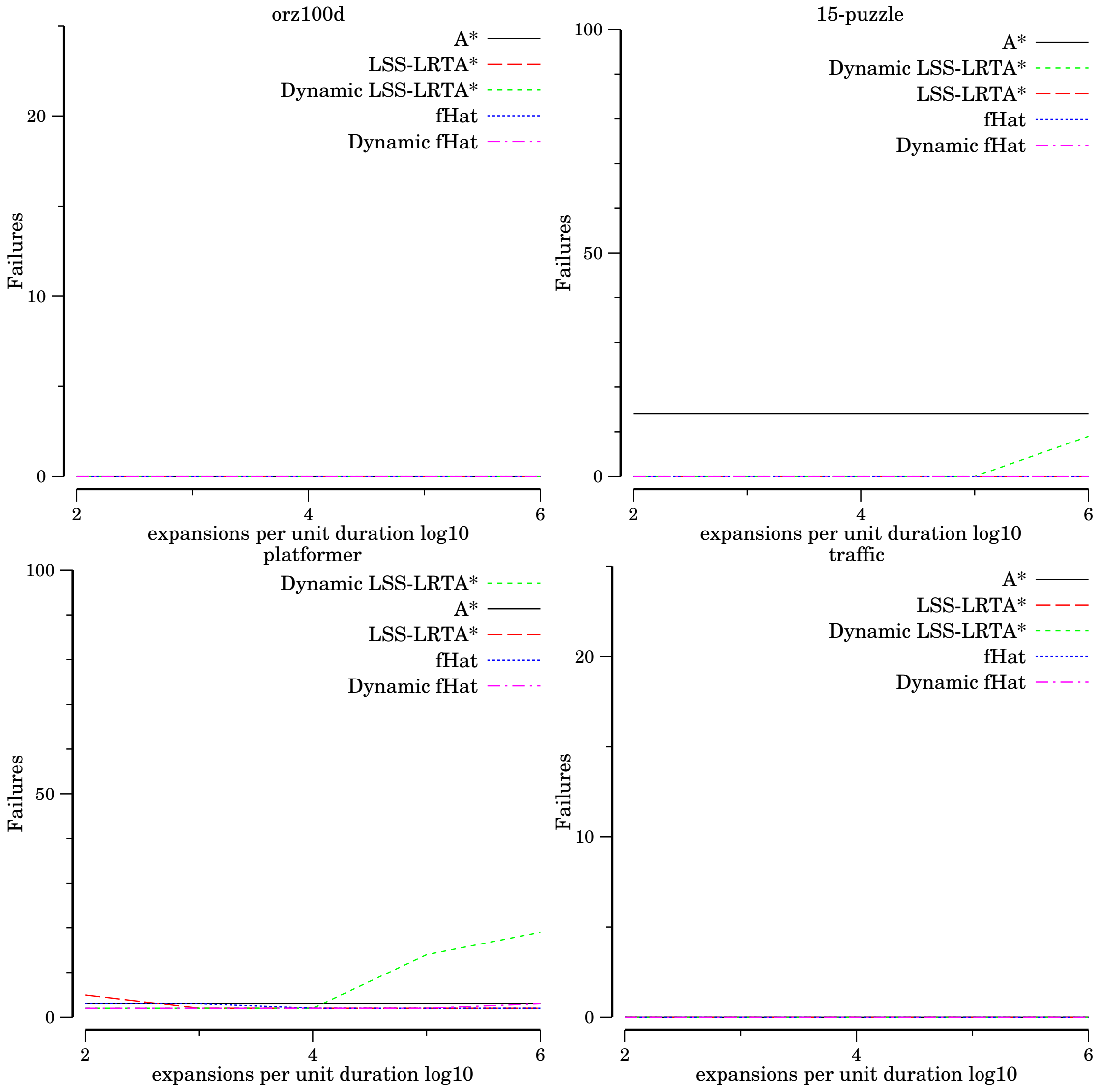


Figure 2-21: Failure rates of the Dynamic-$\hat{f}$ algorithm

failure. This is likely due to the improved search space pruning powers of $\hat{f}$, leading fewer states to be generated and cached.

2.5.4 Shortcomings

In general, $\hat{f}$ does not drastically improve search performance in real-time search outside of memory exhaustion. Dynamic-$\hat{f}$ does embrace the sub-optimal nature of real-time search better than LSS-LRTA* in that it forgoes admissibility for what it expects is a more accurate heuristic, but it is still entirely possible for the agent to be directed toward a dead end by an under informed heuristic.

2.6 DTA*

Previous algorithms have been either offline (in the case of A*) or real-time. DTA* is an online search which, like real-time search, interleaves planning and execution, allowing the agent to reach a goal faster than it might otherwise through offline search. Unlike real-time search, online search is not constrained to a fixed time bound, and can choose freely when to terminate the current iteration of search.. The pseudocode for the DTA* algorithm can be seen in Figure 2-22.

DTA* was designed to use offline training data in order to determine how much search under a given best action would be needed in order to conclude whether or not it is in fact the best action, as well as whether or not the potential improvement is worth the additional search.

2.6.1 Intuition

The intuition behind DTA* is largely the same as the intuition of choosing offline search over online or real-time search. Searches which allow their agents to act before a goal has been found have a significant advantage in that they are not stuck waiting

```
function Choose( State $s$ )
1.    for each child $c$ of $s$
2.        $leaves(c) \leftarrow \{c\}$
3.        $f(c) \leftarrow g(c) + h(c)$
4.    $\alpha \leftarrow$ action of $s$ with lowest $f$
5.    $\beta \leftarrow$ action of $s$ with second lowest $f$
6.    $worth\text{-}continuing \leftarrow$ true
7.    while $worth\text{-}continuing$
8.        for $grain\text{-}size$ steps
9.            $best \leftarrow$ node in $\mathcal{L}_\alpha$ with cost f($\alpha$) and
              lowest probability of exceeding f($\beta$)
10.           if goal($best$) then return $\alpha$
11.           else expand $best$, update $\alpha$, $\beta$,
              and f($\alpha$) accordingly.
12.       find $\mathcal{M}_0$, defined as the set of all leaves $l$ under $\alpha$ such that $f(l) < f(\beta)$
13.       if for some depth $d < depth - horizon$ and some ordered superset $\mathcal{M} \supseteq \mathcal{M}_0$
          $\sum_{x=f'(\beta)}^{\delta-1} \min_{n \in \mathcal{M}} Q_n^d(x) - r|\mathcal{M}|\bar{b}^d \geq 0$ then worth-continuing $\leftarrow$ true
14.       else worth-continuing $\leftarrow$ false
15.   return $\alpha$
```

Figure 2-22: Official pseudocode for DTA* as provided in Do the Right Thing [9]

Term	Description
$\mathcal{L}_\alpha \subset S$	The set of the leaves under α
$\mathcal{M}_0 \subseteq \mathcal{L}_\alpha$	The subset of the leaves in $\mathcal{L}_\alpha$ containing at least all states with estimates lower than that of β
$\mathcal{M} \subseteq \mathcal{L}_\alpha$	An arbitrary superset of $\mathcal{M}_0$
$S^d_{\mathcal{M}}$	The act of expanding all states in $\mathcal{M}$ to depth d
$f'(n) \in \mathbb{R}$	The backed up estimate of n prior to $S^d_{\mathcal{M}}$
$\delta \in \mathbb{R}$	The cost estimate of the lowest cost leaf in $\mathcal{L}_\alpha - \mathcal{M}$
$\sum_{x=f'(\beta)}^{\delta-1} \min_{n \in \mathcal{M}} Q^d_n(x) \in \mathbb{R}$	An upper bound on the benefit of $S^d_{\mathcal{M}}$
$r\lvert\mathcal{M}\rvert\bar{b}^d \in \mathbb{R}$	Estimate of the expected cost of $S^d_{\mathcal{M}}$

Table 2.2: A reference for terms pertinent to the DTA* algorithm

at the start location for long stretches of time. Offline optimal search has a significant advantage as well, however, in that the resulting solution is the best one can hope to achieve. DTA* attempts to determine the likelihood that the path it is following is the same an optimal offline search would return. If it is uncertain as to the answer, and if it suspects that it may be substantially underperforming, it will extend search time in order to more closely approximate an optimal offline search. If the decision is clear, or if the benefit of the additional search is deemed insufficient, DTA* will move the agent forward in a way that is suspected to reduce the overall time spent searching.

2.6.2 Explanation

The algorithm begins with a simple initialization process (lines 1-6) which sets up each of the top level actions and selects the best two as α and β. If it is the first search iteration or additional search has been deemed worthwhile, we continue past line 7 to lines 9-11 where we expand the local search space under α until we find a goal or the user imposed lookahead limit is reached. Note that in line 11 it is possible that α and β swap places. In lines 12-14, we approximate the expected utility of additional search relative to likelihood of β to supersede α as the difference between an upper bound on the estimated benefit and the estimated cost of search. If utility is positive, we continue searching. Otherwise, we terminate search centered around s and move the agent to α. A list of terms used the DTA* pseudocode and their respective meaning is available in Table 2.2.

2.6.3 Evaluation

We evaluated the performance of DTA* relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to previous algorithms; the results are shown in Figures 2-23, 2-24, and 2-25 respectively.

In general, DTA* has a relatively consistent GAT and total solution cost performance across all action duration controls. This is likely because DTA* makes its meta level evaluation using the known ratio of action cost to search time, and will implicitly find a perceived optimal ratio of search iteration length to action duration. This ratio exists only in so far as an increasing cost of search as the LSS expands will lead the algorithm to universally search more when search is cheap and less when it is expensive. This approach yields competitive results for the 15-puzzle (a well studied domain in the context of DTA*) and platformer (when successful) for shorter action durations, but not for longer action durations, as other algorithms tend to see improved performance while DTA*'s tends to remain relatively constant. The poor performance of DTA* on the orz100d domain is likely because of the learning process, which samples heuristic values at random states and associates them with

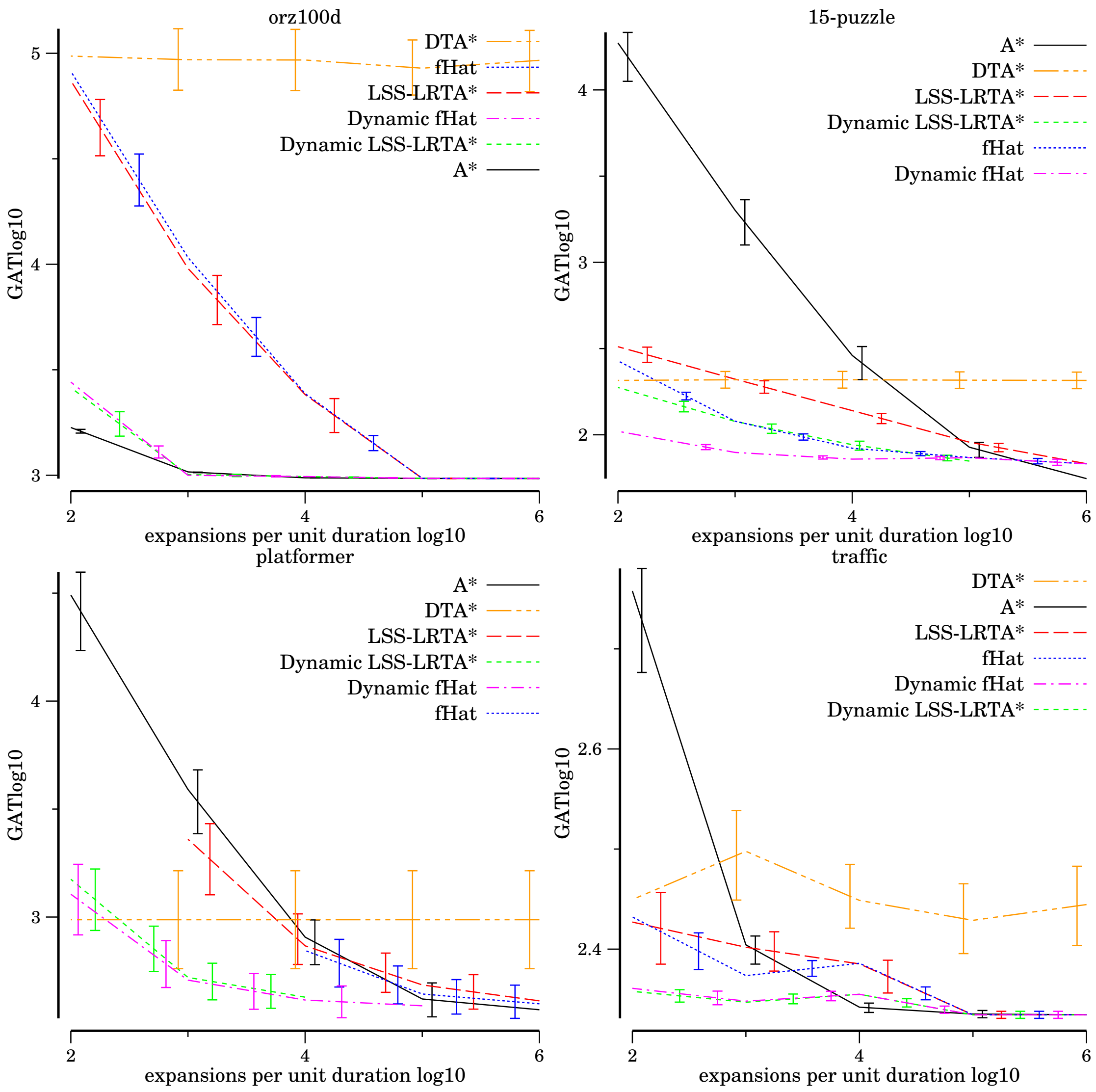


Figure 2-23: GAT performance of the DTA* algorithm

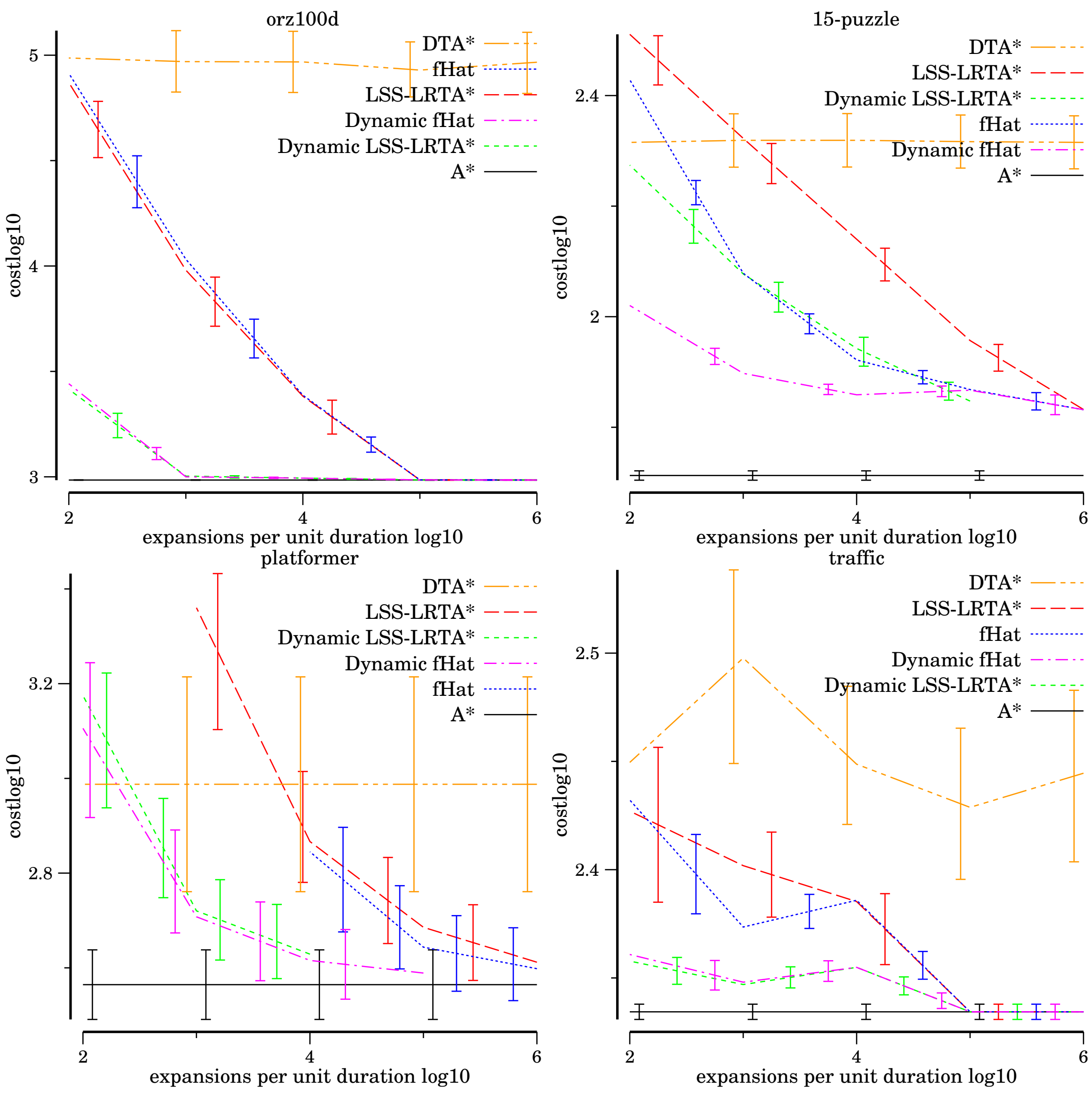


Figure 2-24: Total solution cost performance of the DTA* algorithm

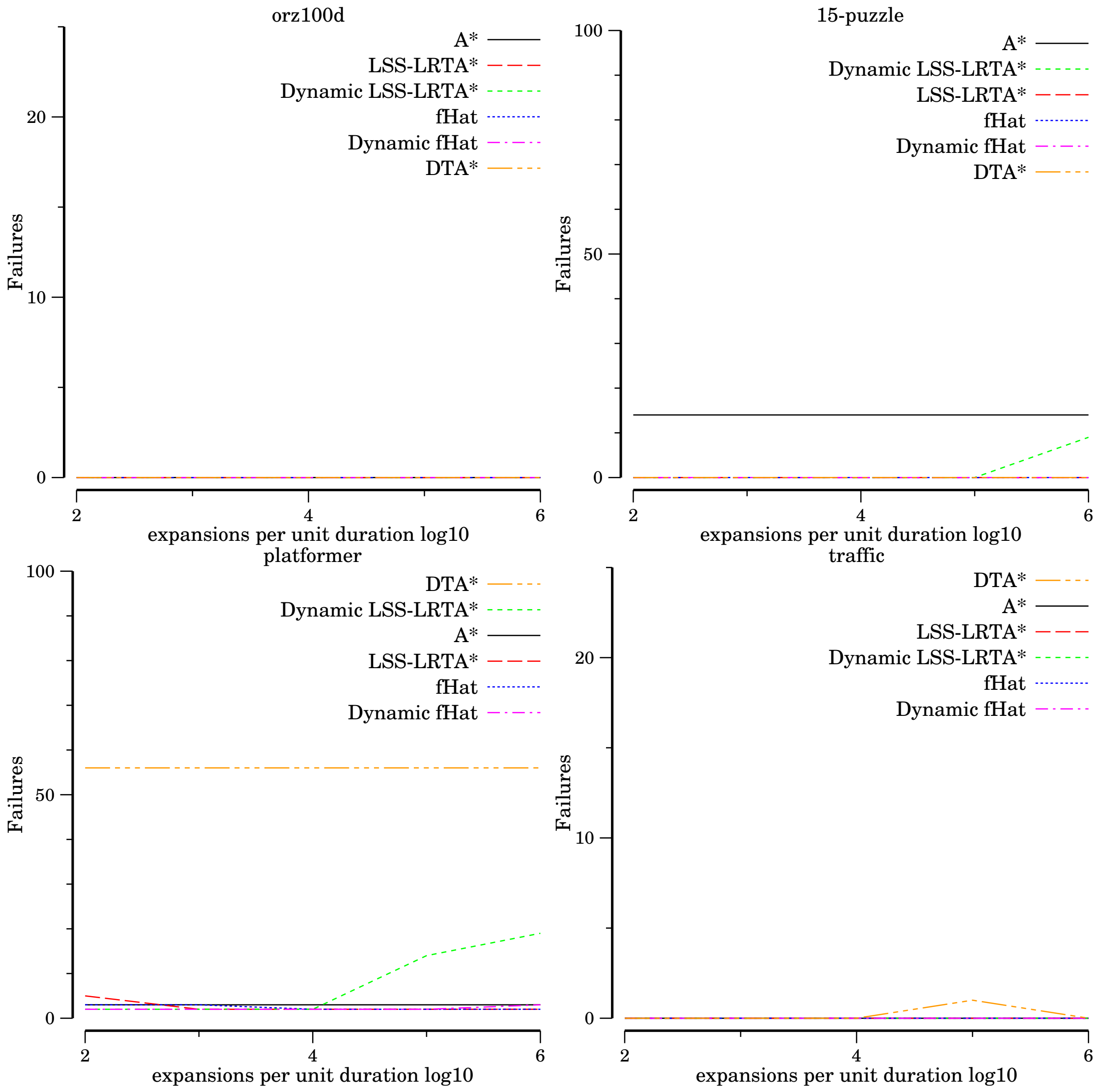


Figure 2-25: Failure rates of the DTA* algorithm

more informed heuristic estimates gathered after short lookahead. This process tends to work well when heuristic error is more uniform, but in cases like orz100d with wide open spaces separated by walls, many states in the open spaces will have their naïve heuristic associated with an equally naïve heuristic if the lookahead is not large enough to encounter an obstacle, biasing the learned values toward optimism. The traffic domain once again yields seemingly strange results in terms of GAT and total solution cost, however this too can likely be attributed to the learning process, as many paths seem promising for short lookaheads but prove unfeasible with additional search. The failure rate for DTA* on the platformer domain is relatively high, which is unexpected given the GAT and total solution cost performance of the successful trials. These failures were largely due timeouts rather than memory exhaustion. This suggests that search was likely directing the agent along a feasible path (not scrubbing), but simply committing too much time to search in antagonistic instances.

2.6.4 Shortcomings

In addition to requiring the collection of training data which is not always possible or practical, DTA* suffers from the assumption that search is being carried out in a tree with disjoint ancestors, and so when applied to a graph, it incurs large amounts of overlapping search. This may not be inherent to the design of the algorithm, but is the standard implementation presented by Russel and Wefald. Additionally, Russel and Wefald themselves note that the offline collected data used in the meta level decision process is inaccurate, as it is gathered using fixed depth lookaheads which are likely to be surpassed during the actual search.

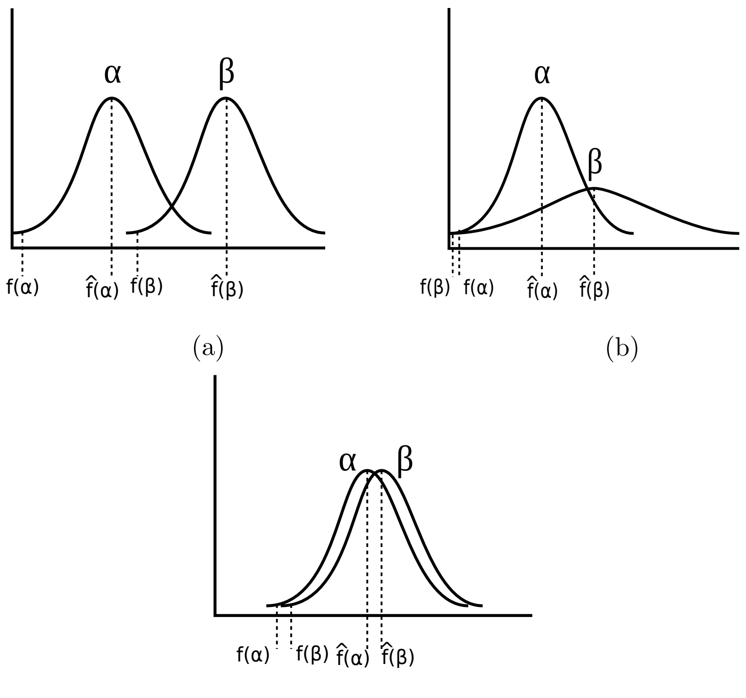


Figure 2-26: Intervals considered by Ms. A*

2.7 Ms. A*

Another metareasoning algorithm in recent development is Ms. A* (E. Burns, S. Kiesel, W. Ruml, submission under review). Ms. A*, much like DTA*, attempts to reason about its certainty in the existing best action, but does so using a simplified approach inspired by interval estimation [10], as shown in Figure 2-27. Unlike previous real-time searches, which always commit to and center search around the best state on the frontier, Ms. A* may choose to cut the queued path short in order to further investigate an area of uncertainty or even forgo an action and center additional search around the agent's current state when possible.

2.7.1 Intuition

If the decision is made to follow the current path until its end, the agent will of course have more time to dedicate to its future decision but it might also commit to a less than favorable action along the way. The algorithm also attempts to recognize what it refers to as identity actions, which are actions which can be taken which do not change the world state. These actions can be taken allowing the current search space to be expanded further. Alternatively, Ms. A* considers committing to only a prefix of the queued path, terminating it short of the frontier, if there is some uncertainty about one of the queued actions versus its best alternative.

The fundamental logic dictating the need for an action/search evaluation to minimize expected total solution cost is sound, but the method of evaluation used for meta level reasoning in Ms. A* is qualitative and overly simplistic. Ms. A* considers three scenarios, depicted in Figure 2-26, for each state s along the trajectory produced by an LSS-LRTA* style procedure. Each scenario is represented as a different possible relative layout of f-value cost estimates and $\hat{f}$-value online corrected cost estimates. The former are computed as per standard practice as the sum of a given states g and h values, while the latter is computed using Jordan Thayer's global error correction [11]. Because of the nature of this online correction, the $\hat{f}$ estimate of any given state is always greater than or equal to the f estimate of the same state. Additionally, $\hat{f}(\alpha_s)$ is by definition less than or equal to $\hat{f}(\beta_s)$. These two restrictions mean that there are three possible relative layouts for the estimates of α_s and β_s. In scenarios a and c, Ms. A* concludes that α_s is superior to β_s with high likelyhood as it wins by both the f and $\hat{f}$ metrics. In either of these cases, we commit to α_s rather than centering the next iteration of search around s. In scenario b, however, we recognize that the variance between f_{β_s} and $\hat{f}_{\beta_s}$ are such that further search might in fact lead to β superseding α as the perceived best action. In simplest terms, Ms. A* will center search around state s if and only if s is the first state along the path to the best state on the frontier such that $f_{\beta_s} < f_{\alpha_s}$.

2.7.2 Explanation

Though its metareasoning practices are inspired by DTA*, Ms. A*'s form is nearly identical to Dynamic $\hat{f}$ with the exception of two additional blocks used for metareasoning. The first (lines 6 and 7) consists of a check for and possible execution of an identity action. In the event that the remaining queued path P is empty, the agent will either have reached the end of its previously queued path or will have been forced to dismiss the remainder due to potential invalidation in the case of dynamic environments (not considered in this thesis). In either case, the agent's current state s lies at the root of the current search space, and so if an identity action exists at that state, Ms. A* determines whether or not there is a significant chance of further search showing favor to β_s. If it does, the identity action is selected. The second block (lines 11-13) is encountered in the event that an identity action is decidedly not practical. In this case, we look along the path for the earliest state past which we have doubts about committing. This prevents the agent from committing too far down a path with uncertain payoff. The selected state is then assigned as the temporary subgoal s', and the search continues centered around it until it is reached by the agent.

2.7.3 Evaluation

We evaluated the performance of Ms. A* relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to previous algorithms; the results are shown in Figures 2-28, 2-29, and 2-30 respectively.

In general, Ms. A* performs on par with Dynamic-$\hat{f}$, performing slightly better on the orz100d domain for shorter unit durations. Ms. A* is approximately as failure resistant as Dynamic-$\hat{f}$, with the exception of the platformer domain where it exhibits

```
function Search( )
1.    while $s \notin S_g$
2.      Perform $BFS(\hat{f})$ for $lookahead$
        expansions on $OPEN$
3.      if $OPEN = \emptyset$
4.        stop
5.      if $P = \emptyset$
6.        if $identity_s \neq \emptyset$ and $f(v_{\beta_s}) < f(v_{\alpha_s})$
7.          $P \leftarrow \{identity_s\}$
8.        else 9.    $s' \leftarrow$ state in $OPEN$ with the lowest $\hat{f}$
10.         Run modified Dijkstra's algorithm on $CLOSED$
11.         For each state $s''$ from $s$ to $s'$ in reverse
12.           if $f(v_{\beta_{s''}}) < f(v_{\alpha_{s''}})$
13.             $s' \leftarrow s''$
14.         $P \leftarrow$ path from $s$ to $s'$
15.     Pop and emit the head of $P$
16.     if edge costs change
17.       $P \leftarrow \emptyset$
18.       $OPEN \leftarrow \{s\}$
```

Figure 2-27: Pseudocode for Ms. A*

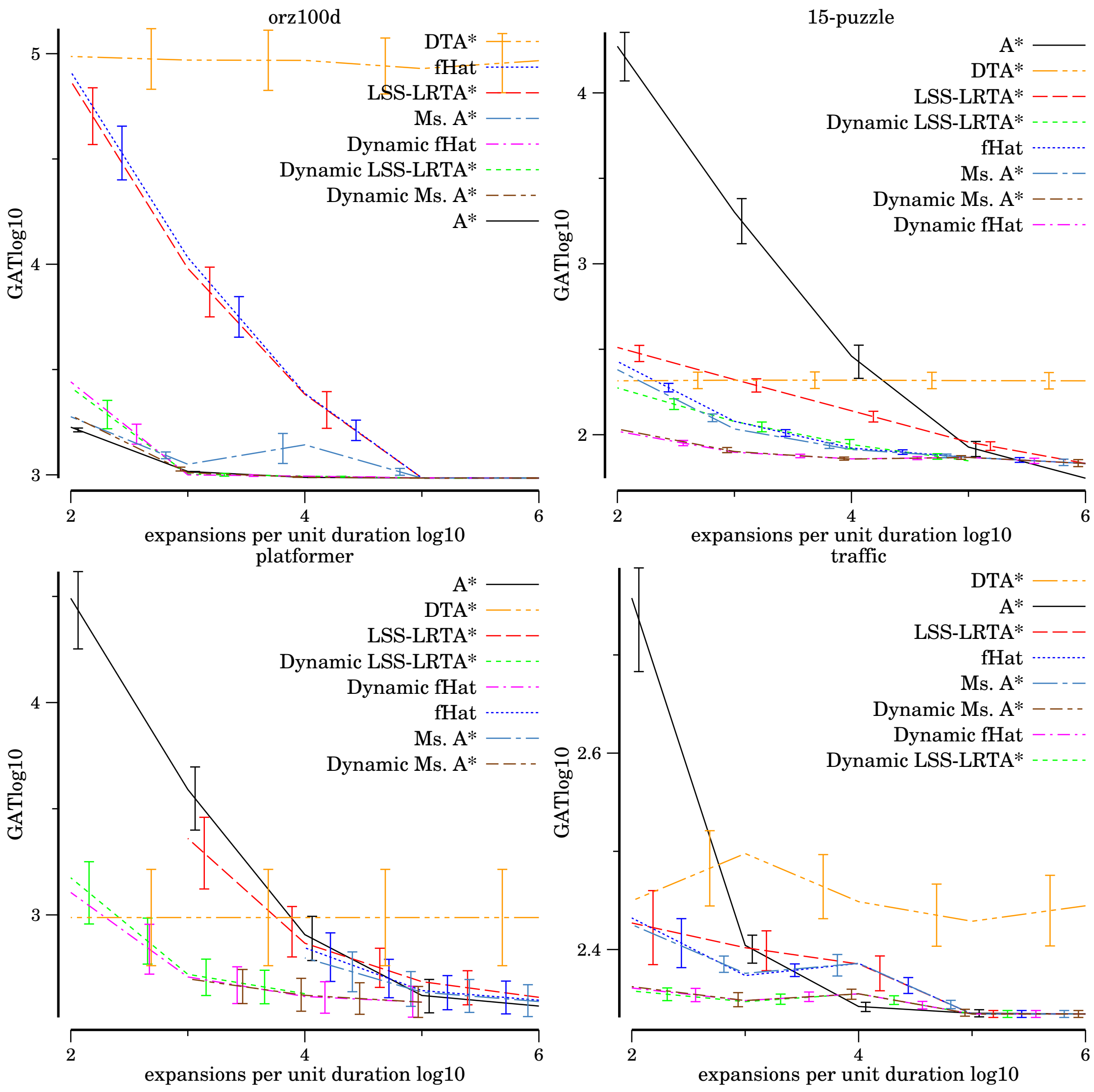


Figure 2-28: GAT performance of the Ms. A* algorithm

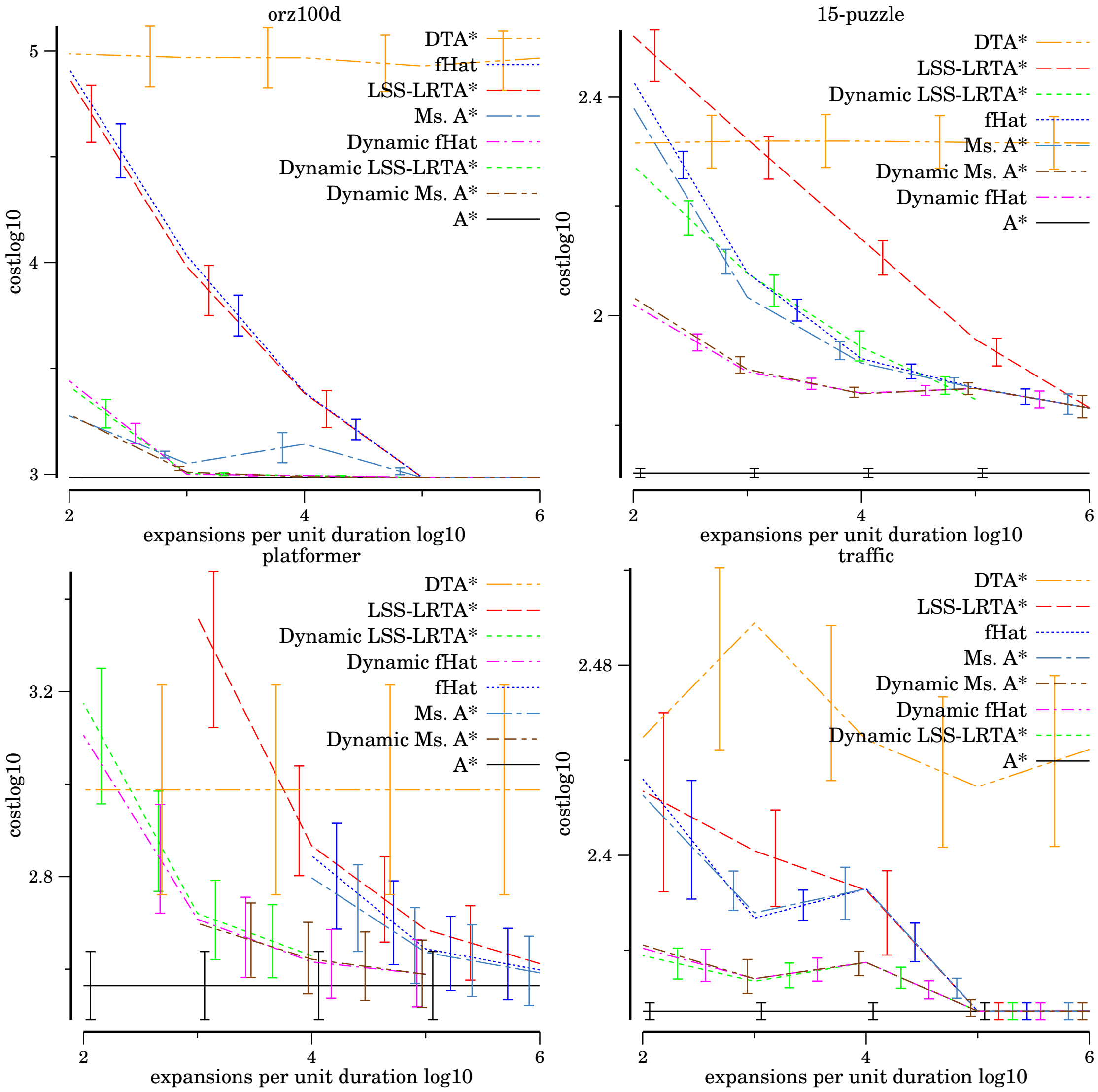


Figure 2-29: Total solution cost performance of the Ms. A* algorithm

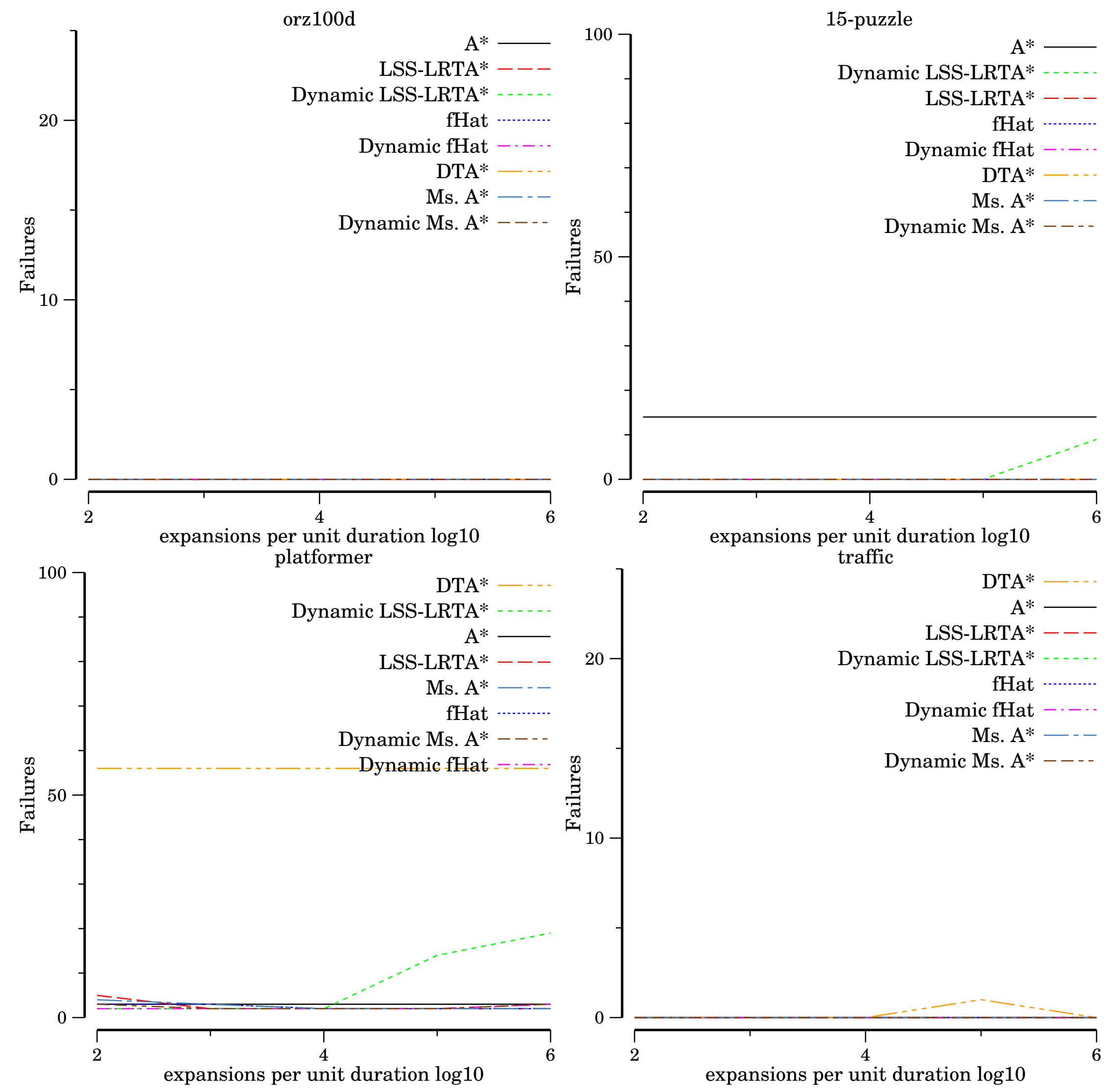


Figure 2-30: Failure rates of the Ms. A* algorithm

a slightly higher failure rate when only 100 states can be expanded per unit duration. Initial speculation may suggest that this failure is due to Ms. A* behaving like A* on an instance where A* failed, however the failure occurred on an instance separate from that which A* failed on. Instead, because Ms. A* failed on an instance where both A* and Dynamic-$\hat{f}$ succeeded, Ms. A* must have been mislead by the meta level decision process in some way.

2.7.4 Shortcomings

While similar to the approach we propose in the next chapter, Ms. A* has a heavily oversimplified perception of the meta level decision. In addition to making a qualitative rather than quantitative assessment of the value of search based on overlapping f-$\hat{f}$ intervals, the cost of search is not considered at all. Fundamentally, the meta level decision process comes down to a simple check of whether or not $f(\beta) < f(\alpha)$, with no regard to any other factors at play.

2.8 Summary and Perspective

In this chapter we discussed several previously developed algorithms, detailing the design, intuition, and shortcomings of each. We presented the framework with which we evaluate the performance of the coming algorithms and we have used it to compare the performance of said existing works. From here, we will attempt to address some of the shortcomings highlighted in this chapter and further compare the resulting algorithms using the defined framework, namely the issue of optimistically following the heuristic along what appears superficially to be the optimal path despite an inherent expectation of suboptimality in real-time search.

Chapter 3

Identity Action Selection

As A* has shown empirically by dominating the GAT of its competitors for short unit durations on the orz100d domain, it is occasionally desirable to forgo progressing the agent toward the goal for one or more search iterations in order to improve certainty about the correctness of the choice being made. We now present a principled means with which to make that meta level decision. We call the presented algorithm and its variant MORTS (Metareasoning Online Real-Time Search).

3.1 Framework

The general design of the identity action variant of the MORTS algorithm is shown in Figure 3-1.

This algorithm behaves almost identically to Ms. A* with two key differences. First, for the sake of simplicity of analysis, the algorithm does not consider cutting the queued path short. Instead, only identity action and full path commitment are considered. This prevents performance impact from path prefixing from obfuscating the impact of identity actions and vice versa. The second difference is that in the meta level decision process in line 7, we replace the overly simplistic qualitative evaluation used in Ms. A* with a more sophisticated quantitative analysis, described in this chapter.

```
function Search( )
1.    while $s \notin S_g$
2.       Perform $BFS(\hat{f})$ for $lookahead$
         expansions on $OPEN$
3.       if $OPEN = \emptyset$
4.          stop
5.       if $P = \emptyset$
6.          $s' \leftarrow$ state in $OPEN$ with the lowest $\hat{f}$
7.          if $identity_s$ and $Benefit(s) > c(identity_s)$
8.             $P \leftarrow identity_s$
9.          else 10.   Run modified Dijkstra's algorithm on $CLOSED$
11.            $P \leftarrow$ path from $s$ to $s'$
12.            $OPEN \leftarrow \{s\}$
13.            $CLOSED \leftarrow \{s\}$
14.      Pop and emit the head of $P$
15.      if edge costs change
16.         $P \leftarrow \emptyset$
17.         $OPEN \leftarrow \{s\}$
18.         $CLOSED \leftarrow \{s\}$
```

Figure 3-1: Pseudocode for the Generic MORTS Ident algorithm

3.2 Intuition

If our goal is to optimize the expected goal achievement time, it makes sense that we might wish to forgo emitting an action in order to gain insight into which available action bodes more favorably, as Ms. A* does. While Ms. A*'s approach to this analysis has been shown empirically to cause some small improvement in GAT over Dynamic-$\hat{f}$ in certain domains for short unit durations, a more sophisticated approach may yield better results.

Consider the potential outcomes of taking an identity action. The utility of an identity action may be quantified in terms of a known cost $c(identity_s)$ representing the time spent not moving the agent toward the goal and the probabilistic effect of β superseding α as best top level action. Let us first denote the difference in true cost to go as $\Delta^*_{\alpha\beta}$, defined in Equation 3.1.

$$\Delta^*_{\alpha\beta} = f^*(\beta) - f^*(\alpha) \tag{3.1}$$

The utility of search u can then be expressed as the difference between the still unknown value of $\Delta^*_{\alpha\beta}$ and the known cost $c(identity_s)$, expressed in Equation 3.2.

$$u = \Delta^*_{\alpha\beta} - c(identity_s) \tag{3.2}$$

It should be noted that $f^*(\beta)$ may actually be lower than $f^*(\alpha)$, in which case the identity action has not only incurred the expenditure of time during which the agent is not progressing toward the goal, but has introduced a false positive or type 1 error as well, sending the agent along a path which actually has a higher true total solution cost than it would have otherwise incurred by following α. With this in mind, there are several distinct utility outcomes that may result from the meta level decision, outlined in Table 3.1.

	No Search	Select α	Select β
$f^*(\alpha) < f^*(\beta)$	0	$-c(identity_s)$	$-\|\Delta^*_{\alpha\beta}\| - c(identity_s)$
$f^*(\alpha) \geq f^*(\beta)$	$-\|\Delta^*_{\alpha\beta}\|$	$-\|\Delta^*_{\alpha\beta}\| - c(identity_s)$	$\|\Delta^*_{\alpha\beta}\| - c(identity_s)$

Table 3.1: A table of potential utility outcomes resulting from additional search

Note that if no additional search is employed, α is selected regardless of its efficacy, but no additional cost is incurred. Otherwise, a cost is incurred in the form of $c(identity_s)$, and total solution cost may be changed, either positively or negatively, if and only if β supersedes α at the end of the additional search. Considering this, the probabilistic utility of search u^p_s is as shown in Equation 3.3.

$$u^p_s = |\Delta^*_{\alpha\beta}| * (P(\beta|\alpha \geq \beta) - P(\alpha|\alpha \geq \beta) - P(\beta|\alpha < \beta)) - c(identity_s) \tag{3.3}$$

We should then only search if the probabilistic utility of searching u^p_s exceeds the probabilistic utility of acting without searching, shown in Equation 3.4.

$$u^p_a = -|\Delta^*_{\alpha\beta}| * P(\alpha \geq \beta) \tag{3.4}$$

Unfortunately, as the true value of $\Delta^*_{\alpha\beta}$ is outside the scope of our knowledge at the time of the meta level decision process, we must find some estimate of utility instead. In this chapter we present, among other things, two options for estimating said utility.

3.3 Assumptions

There are a number of assumptions made in the design of MORTS.

First and most fundamentally, we assume that there exist conditions under which it is preferable to continue searching rather than acting, and these conditions are knowable or at least estimable.

The first half of this assumption seems fairly intuitive. If faced with a scenario where executing actions is a relatively lengthy and expensive process, and expanding states is relatively quick and cheap, it makes sense that there could be merit to waiting on additional search before committing to an incorrect action. The second half of the assumption, however, is less clear as it is not immediately obvious how to determine whether more search is likely to result in a better outcome and with what likelihood.

Another assumption which is important to consider when rationalizing the utility estimation process presented in this chapter is that we will not encounter type 1 error as a result of additional search. That is to say, if we choose to perform additional search and β supersedes α as best top level action by the time we choose to emit an action, it is because β is in fact better than α. This assumption is important because it is impossible to account for fully, as to do so would require that we know or have some estimate of the true value of $\Delta^*_{\alpha\beta}$ when of course such knowledge would preclude us from searching (as we would then assume β is in fact worse than α and choose not to search). The justification for this assumption lies in the fact that we are already estimating the certainty of our beliefs. If our estimates of certainty are at all reasonable, we make the right meta level decision more often than not, increasing the likelihood of search when β is better than α and decreasing the likelihood of terminating search after selecting an inferior action.

3.4 Cost of search

While in reality additional search carries with it the risk of type 1 error and its associated cost, we have already made the assumption that such an error is no more likely to occur through additional search than through forgoing it. With this simplification, the cost of search is simply the number of unit action durations during which the agent is not progressing along a (supposed) path to a goal. This cost is only incurred in the event of identity actions.

3.5 Benefit of search

The benefit of search is, as previously established in Table 3.1, dependent on whether the search causes β to supersede α as well as whether $f^*(\beta)$ is actually better than $f^*(\alpha)$ and by how much. Because we do not know either of these values at the time of the meta level search decision, we must instead estimate them. For this we make a normal distribution (called the belief distribution) for both $f^*(\alpha)$ and $f^*(\beta)$ denoting the expected probability of said variable possessing a given value. There are a number of parameters which may be reasonably selected for these distributions, but in this paper we constrain them according to the following assumptions:

The mean of the belief distribution for $f^*(s)$ is some online corrected (inflated) estimate of $f^*(s)$ no less than $f(s)$. This assumption is based on the increased accuracy of the online corrected estimate. The goal of the online correction process is to make the corrected metric more accurate than the base metric f, and so if the online corrected metric is successful, it should be the closest estimate of f^* that we have available.

The variance of the belief distribution for $f^*(s)$ is equal to the squared difference of the mean and $f(s)$. Because this difference represents an estimate inflation gathered by uncertain means, and which is continually changing throughout search, we assume that it may reasonably change equally as much in either direction during the remaining search. Additionally, if f is admissible, we know with certainty that the true total solution cost f^* will not be less than f, making it a good lower bound.

Because the variance of the distribution is dependent entirely on the difference between the mean and a fixed value, we need only to control the mean of the distribution. We will now present two approximations of $f^*(s)$ which have been employed as the means of these distributions.

3.5.1 Global Error Estimate

The first metric tested was the same $\hat{f}$ estimate by which we have been ordering search. In forming this estimate, we have collected a global average of the one-step heuristic error experienced throughout search ϵ, multiplied this average by the expected number of steps remaining to the goal $d(s)$ and added this error to $f(s)$. This means that if the mean $\hat{f}(s)$ is simply $f(s) + \epsilon * d(s)$, the variance is $(\epsilon * d(s))^2$.

Intuition

The intuition behind this approach should be straightforward. The global error ϵ serves as a correction to f in hopes of making a better approximation of f^*. If we trust $\hat{f}$ to accomplish this task, it makes sense that we should use it as the mean of our belief distribution, as this is where we suspect f^* is likely to lie with highest probability.

Shortcomings

There is one apparent shortcoming in using $\hat{f}$ as the mean, which is that states with similar $\hat{f}$ values generally have similar f values and thus belief distributions as well. Because ϵ is calculated as a rolling average through search, and because d estimates for many domains are closely correlated with their h counterparts, and because states with similar $\hat{f}$ values are generally generated in tight succession (as search is sorted on $\hat{f}$), states with similar $\hat{f}$ estimates will generally have similar error, and thus f estimates, as well. While this is not necessarily problematic, it means that distributions for significantly different states may not be sufficiently differentiable from each other. This in turn means that the identity action decision relies primarily on the presence of a large global ϵ.

3.6 Path-Based Error Estimate

Another metric which has shown promising results is a more history dependent path-based error model. For this model, we still order search by $\hat{f}$ as that has been shown empirically to perform best at pruning search [11], but instead of using a global error model to compute the mean of the belief distributions, we take the average single step error along the path from the root of search (the agent's current location) to the best leaf of a given top level action. This value, call it ϵ_s is once again multiplied with $d(s)$ and added to $f(s)$ to form the mean of the distribution. The advantage of this model over the global error model is that the mean of a given belief distribution is not influenced by less directly related states encountered throughout search. We refer to the variant of the MORTS algorithms using this error model as the Hybrid variants, as they hybridize the use of online correction by sorting on a global error estimate and using a path-based estimate for the formation of the belief distributions.

Intuition

The intuition for this approach is that, while the globally learned $\hat{f}$ metric may outperform other known online correction techniques in terms of sorting the open list and pruning search, it may not be as accurate as an actual estimate of f^*. By still ordering search by $\hat{f}$ and using this path-based metric for distribution computation, we have the advantage of both pruned search and substantial differentiation between the belief distributions of dissimilar states.

In Table 3.2, we have collected some samples in each of the test domains showing distributions of f^* and average belief distributions for each of the error models for

Domain	$\hat{f}$ Range	Global	Path	f^*
orz100d	150-155	$\mu = 154$, $\sigma = 2$	$\mu = 287$, $\sigma = 135$	$\mu = 396$, $\sigma = 97$
	250-255	$\mu = 255$, $\sigma = 3$	$\mu = 528$, $\sigma = 275$	$\mu = 496$, $\sigma = 103$
	350-355	$\mu = 356$, $\sigma = 4$	$\mu = 1059$, $\sigma = 707$	$\mu = 769$, $\sigma = 78$
15-puzzle	10-12	$\mu = 13$, $\sigma = 1$	$\mu = 17$, $\sigma = 5$	$\mu = 23$, $\sigma = 6$
	30-32	$\mu = 50$, $\sigma = 19$	$\mu = 65$, $\sigma = 34$	$\mu = 26$, $\sigma = 5$
	50-52	$\mu = 82$, $\sigma = 31$	$\mu = 110$, $\sigma = 59$	$\mu = 35$, $\sigma = 3$
platformer	20-25	$\mu = 26$, $\sigma = 1$	$\mu = 28$, $\sigma = 3$	$\mu = 48$, $\sigma = 10$
	50-55	$\mu = 62$, $\sigma = 8$	$\mu = 67$, $\sigma = 14$	$\mu = 64$, $\sigma = 6$
	80-85	$\mu = 87$, $\sigma = 5$	$\mu = 98$, $\sigma = 16$	$\mu = 82$, $\sigma = 8$

Table 3.2: Sample distributions comparing the accuracy of each error model in the tested domains (excluding traffic)

small ranges of $\hat{f}$ values. Samples were taken from single randomly selected instances for each of the tested domains, except for traffic which exhausted memory before samples could finish being taken. A brief qualitative analysis of this data suggests that the path-based error model is, on average, inflated more than the global $\hat{f}$ estimate. This seems to better approximate f^* in many cases where the global model is too optimistic, but only exacerbates the issue when the global model overestimates. The average accuracy of these metrics is only one small piece to consider in their applicability for forming belief distributions, however it does provide some potentially useful insight.

Shortcomings

As we will show in the evaluation section, while generally performing as well as or better than its competitors, this approach can have some detriment to the performance of search in some domains for short action durations.

3.6.1 Search Distributions

After a belief distribution has been formed for each top level action, it can be converted to narrower distributions around the same mean which represent the expectation of where each $\hat{f}$ value lies after an iteration of search. We call these the search distributions, and they have the same means as the belief distributions but smaller variances, as the same single step error is applied to a distance less than $d(s)$. Note that the means of the belief distributions in both of the proposed methods estimate the probability of f^*, or where $\hat{f}$ will rest after d steps toward the goal. We most likely do not search that far toward the goal; instead we estimate how many steps toward the goal search will progress. To do this, we make use of a metric called expansion delay, introduced by Austin Dionne [12]. The expansion delay e_s for any given state s is the number of expansions which take place between its generation and its expansion. $\bar{e}^s_{s'}$ is used to denote the average expansion delay for all states along the best path from s to s'. Given $\bar{e}^s_{s'}$ for both α and β for the most recent search iteration (where s' is current best leaf state of the respective top level action, and s is the first state along the best path from s_{agent} to s' expanded during the current iteration of search) and a lookahead limit l for the next iteration of search, we can estimate the average number of steps an iteration of search will progress along all paths toward a goal as $\frac{l}{e}$. Earlier, we formed the variance of the belief distributions by multiplying a single step error estimate by the estimate of distance remaining to the goal. With this in mind, we multiply the variance of each of the belief distributions by $\frac{\frac{l}{e}}{d(s)}$ in order to estimate where the mean will lie after one iteration of search, rather than after searching all the way to the goal.

3.6.2 Computation of benefit

Once the search distributions have been created for each of the top level actions, there remains the issue of using them to compute the expected benefit of search. To do this, we perform an asymmetrical double integration over the two distributions, defined in Equation 3.5.

$$\mathcal{B}^{\alpha}_{\beta} = \int\limits_{x_\alpha}\int\limits_{x_\beta} (\mathit{if}\ x_\alpha \leq x_\beta, 0, \mathit{else}\ x_\alpha - x_\beta) * P(x_\beta)dx_\beta * P(x_\alpha)dx_\alpha \tag{3.5}$$

where $\mathcal{B}^{\alpha}_{\beta}$ denotes the benefit of search centered around the parent of α and β, and x_α and x_β refer to some value over a selected range along the search distributions for α and β respectively. In this research we bind this range by two standard deviations out from the mean of the respective distribution.

In simplest terms, this Equation 3.5 iterates over all possible combinations of $\hat{f}(\alpha)$ and $\hat{f}(\beta)$ and accumulates the probabilistic benefit of said combination, equal to the product of the outcome's benefit in cost reduction and its probability according to the search distributions. The integration is asymmetrical (we return no benefit rather than negative benefit in the case that α beats β) because we assume that if α is in fact better, we end up selecting α which yields no benefit from the search.

3.7 Evaluation

We evaluated the performance of the MORTS-Ident variants relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to previous algorithms; the results are shown in Figures 3-2, 3-3, and 3-4 respectively. For the purposes of minimizing clutter, Static LSS-LRTA*, Static $\hat{f}$, Static Ms. A*, and DTA* have been removed from future plots, as they are all largely dominated by other algorithms. Additionally, GAT plots with the removal of A* and static variants

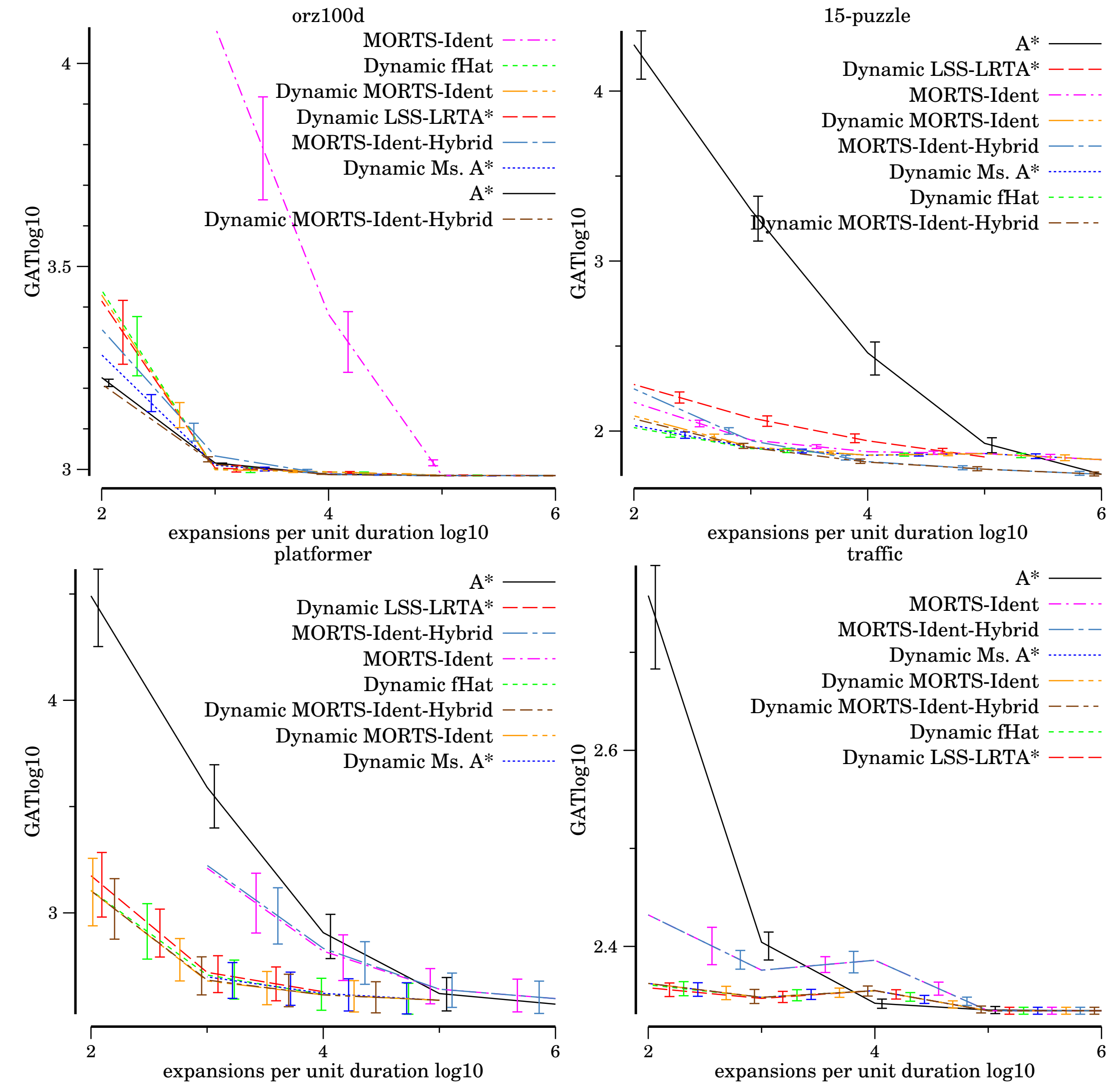


Figure 3-2: GAT performance of the MORTS-Ident algorithm variants

are shown in Figure 3-5 in order to highlight the difference in performance between the more competitve real-time algorithms.

In general, both variants of the MORTS-Ident algorithm perform relatively well, on par with competitors. It is worth noting that the Static paradigm variants display competitive performance against other algortihms using the Dynamic paradigm, with the exception of the global error variant's under-performance in the orz100d. In the platformer and traffic domains, there is not much difference in the performance of

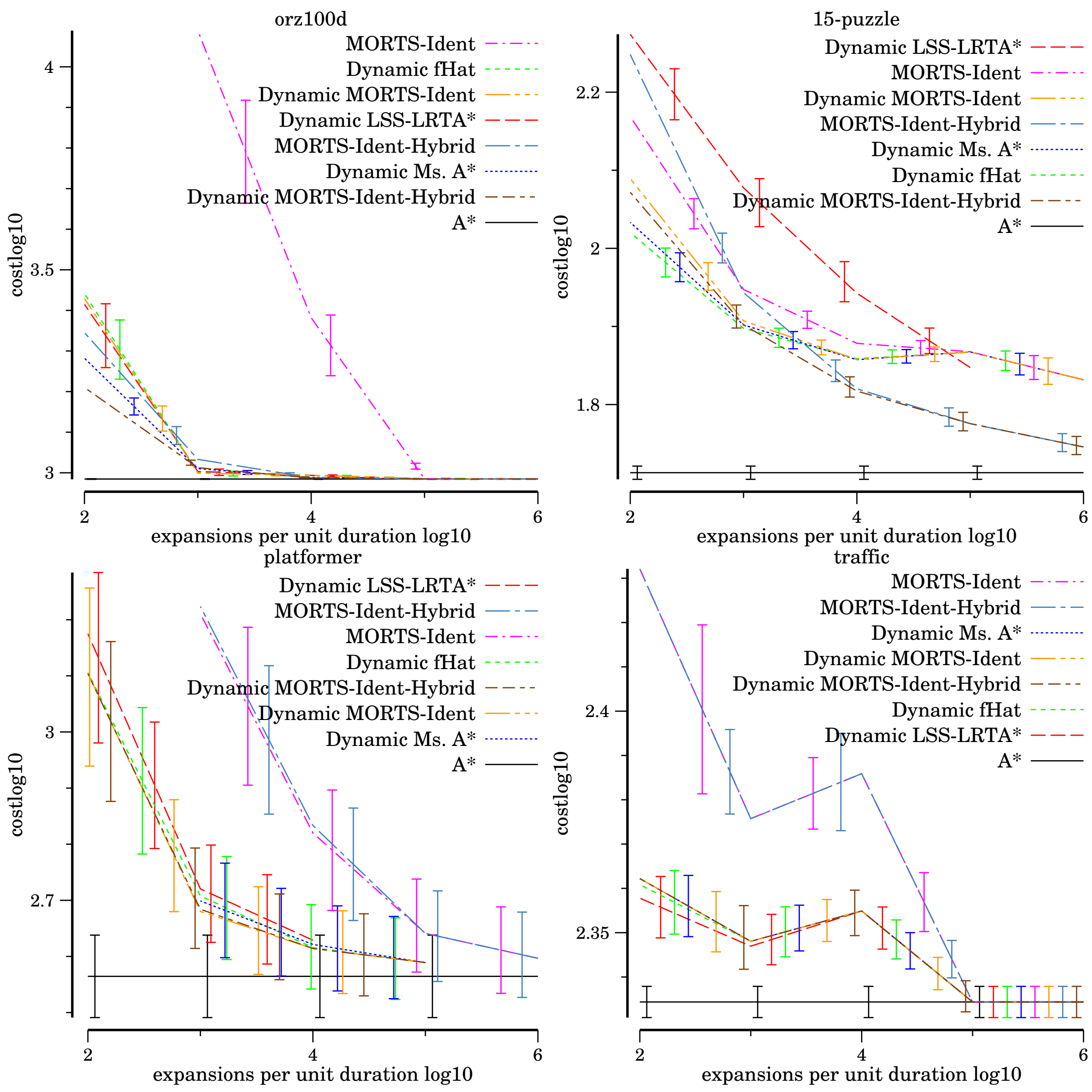
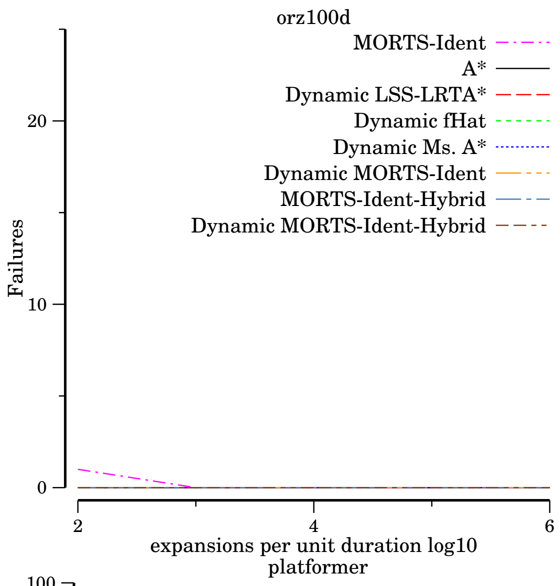
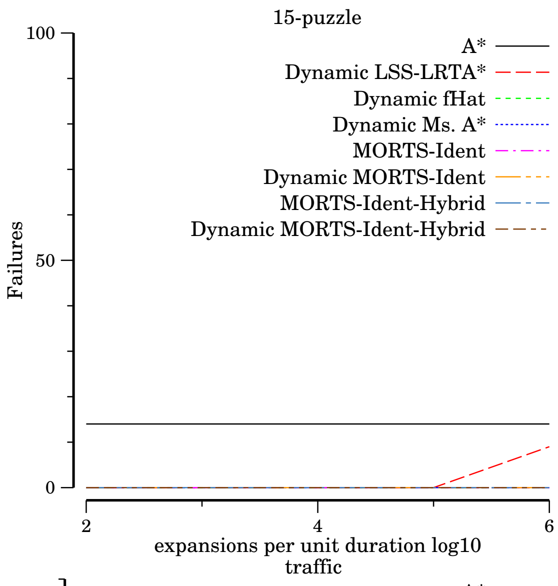
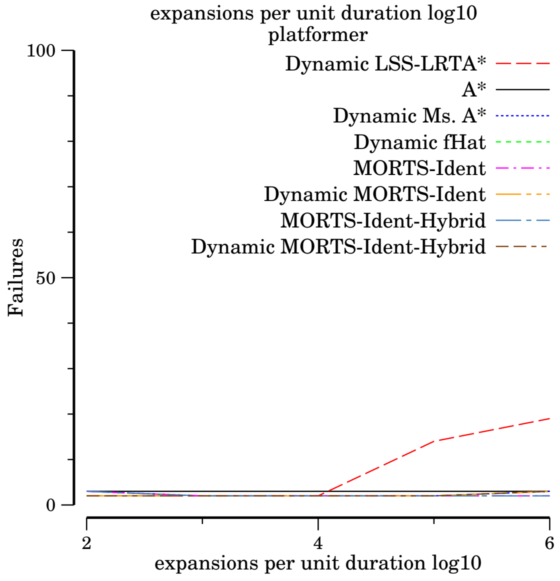


Figure 3-3: Total solution cost performance of the MORTS-Ident algorithm variants







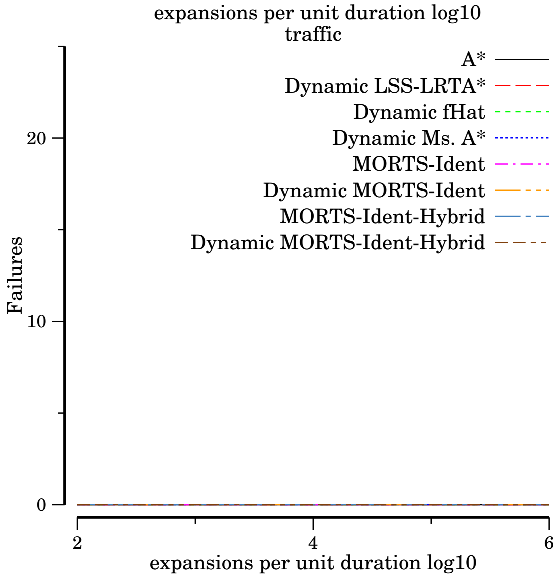


Figure 3-4: Failure rates of the MORTS-Ident variants

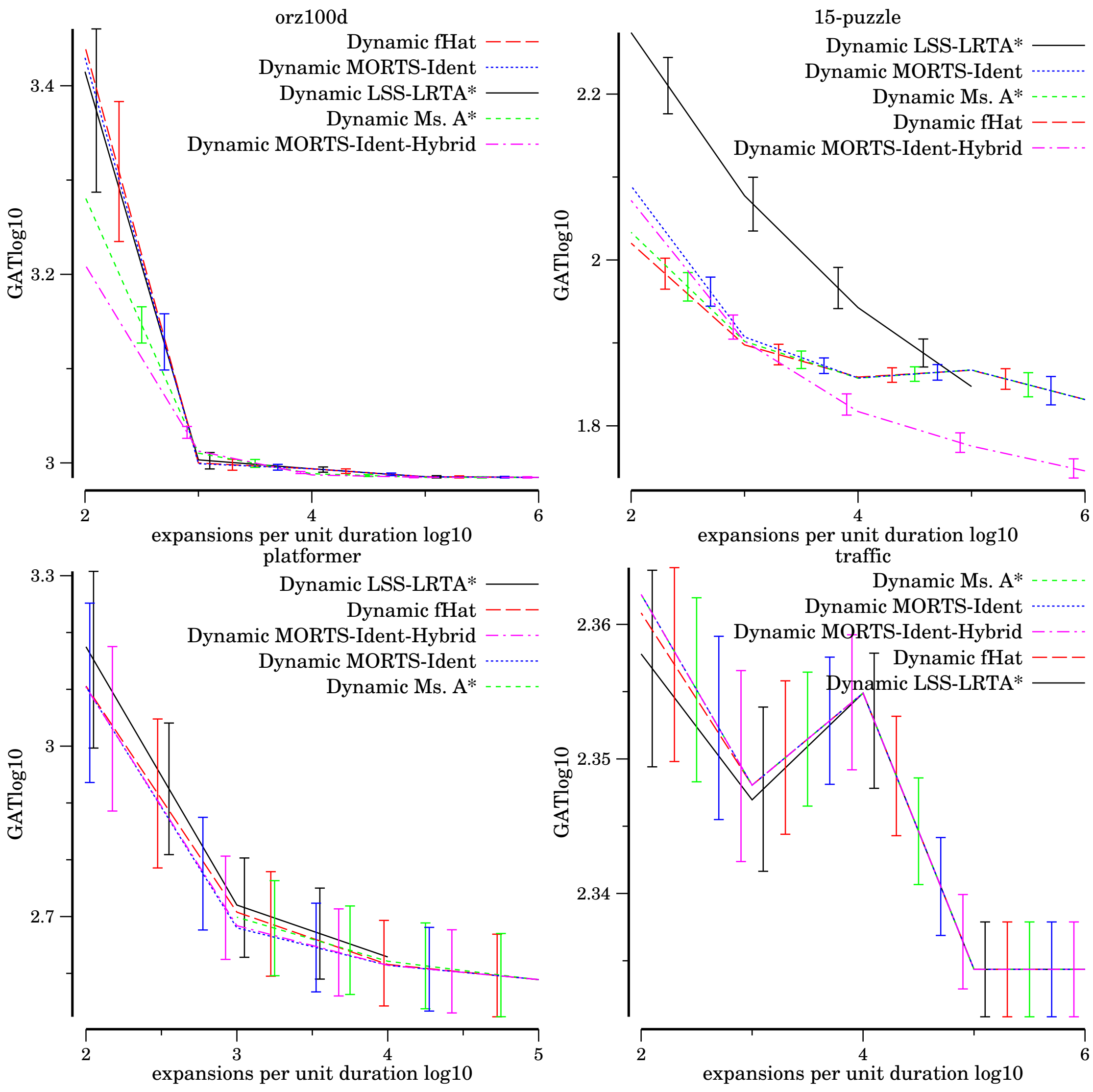


Figure 3-5: GAT performance of the MORTS-Ident variants against competitve RTS algorithms

either of these variants and their counterparts. In the orz100d domain the hybrid variant significantly outperforms competitors for the shortest action durations, and is in fact the first algorithm of those presented to outperform the GAT of A* in this case. In the 15-puzzle domain, both variants are beaten by Dynamic-$\hat{f}$ for the shortest possible action durations, but the global error model variant quickly converges, while the hybrid variant quickly surpasses its competitors. This brief period in which both variants are outperformed by Dynamic-$\hat{f}$ implies identity actions which either resulted in type 1 error or did not otherwise make up for the cost of search. This may suggest that there is some small modification which, through the reduction of non-beneficial identity actions, might allow the hybrid variant to universally outperform its competitors.

Chapter 4

Path Prefixing

In addition to the question of when to take identity actions, we should consider whether it is worthwhile to cut a queued trajectory short so that we can more thoroughly investigate an intermediate state. This problem is distinct from the problem of identity action selection in that there is no explicit cost associated with the search and because we have a finite constraint on the number of action durations available for the search. There is also an implicit but difficult to quantify cost in early termination of a queued path when dealing with dynamic lookaheads, as a shorter path results in a smaller local search space in the next iteration.

4.1 Framework

The general design of the path prefix variant of the MORTS algorithm is shown in Figure 4-1.

Once again the algorithm follows Ms. A* closely with the exception of the evaluation in line 8 in which we perform a quantitative rather than qualitative analysis (previously described), and we only consider half of the meta level decision process. This time, however, we forgo the use of identity actions and instead only consider path prefixing.

```
function Search( )
1.      while $s \notin S_g$
2.         Perform $BFS(\hat{f})$ for $lookahead$
           expansions on $OPEN$
3.         if $OPEN = \emptyset$
4.            stop
5.         if $P = \emptyset$
6.            $s' \leftarrow$ state in $OPEN$ with the lowest $\hat{f}$
7.            Run modified Dijkstra's algorithm on $CLOSED$
8.            For each state $s''$ from $s$ to $s'$ in reverse
9.               if $Benefit(s'' > 0$
10.                 $s' \leftarrow s''$
11.           $P \leftarrow$ path from $s$ to $s'$
12.              $OPEN \leftarrow \{s\}$
13.              $CLOSED \leftarrow \{s\}$
14.        Pop and emit the head of $P$
15.        if edge costs change
16.           $P \leftarrow \emptyset$
17.           $OPEN \leftarrow \{s\}$
18.           $CLOSED \leftarrow \{s\}$
```

Figure 4-1: Pseudocode for the Generic MORTS Prefix algorithm

4.1.1 Intuition

Much like the prefix commitment question posed by Ms. A*, the path prefix variant of MORTS is concerned with finding the earliest point along the queued path for which it is uncertain about the superiority of the selected action. If at some intermediate state s'' along the queued path from s_{agent} to some state s' we have chosen action $\alpha_{s''}$, but action $\beta_{s''}$ exists and leads to a leaf on the frontier which may compete with s', we may want to consider leaving the option open of taking $\beta_{s''}$. There is, of course, an implicit trade taking place here as the path will be shortened and thus so will the next iteration of search, but because this cost is implicit, difficult to quantify, and asymptotically recovered as the duration of future iterations and expansion delay both increase, we ignore it.

4.1.2 Evaluation

We evaluated the performance of the Dynamic MORTS-Prefix variants relative to goal achievement time, total solution cost, and failure rate across the test domains as compared to previous algorithms; the results are shown in Figures 4-2, 4-4, and 4-6 respectively. Similarly, the performance of the static variants is shown in Figures 4-3, 4-5, and 4-7. The static and dynamic variants have been separated into different plots as the algorithm has significantly different behavior for each, as explained in evaluation. In order to minimize clutter and to highlight the performance differences among competitive algorithms, A* has been removed from these plots.

In general, the variants of MORTS-Prefix behave approximately the same, suggesting that there is some distinct difference between the prefix and identity action problems which allows the hybrid variant of MORTS-Ident to significantly outperform its counterpart. Both variants of MORTS-Prefix tend to perform worse than their competitors when following the Dynamic lookahead paradigm, but better when fol-

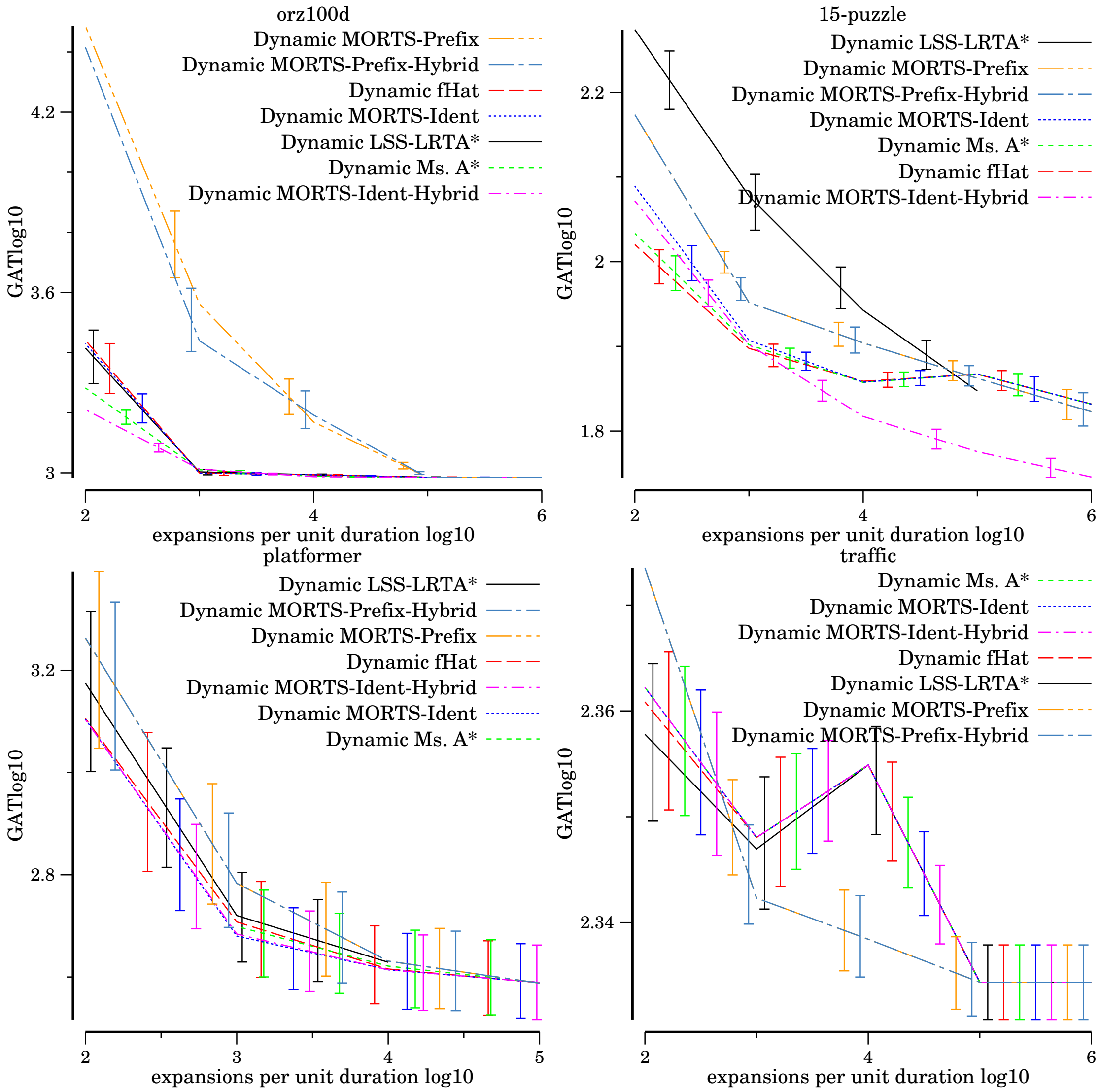


Figure 4-2: GAT performance of the Dynamic MORTS-Prefix algorithm variants

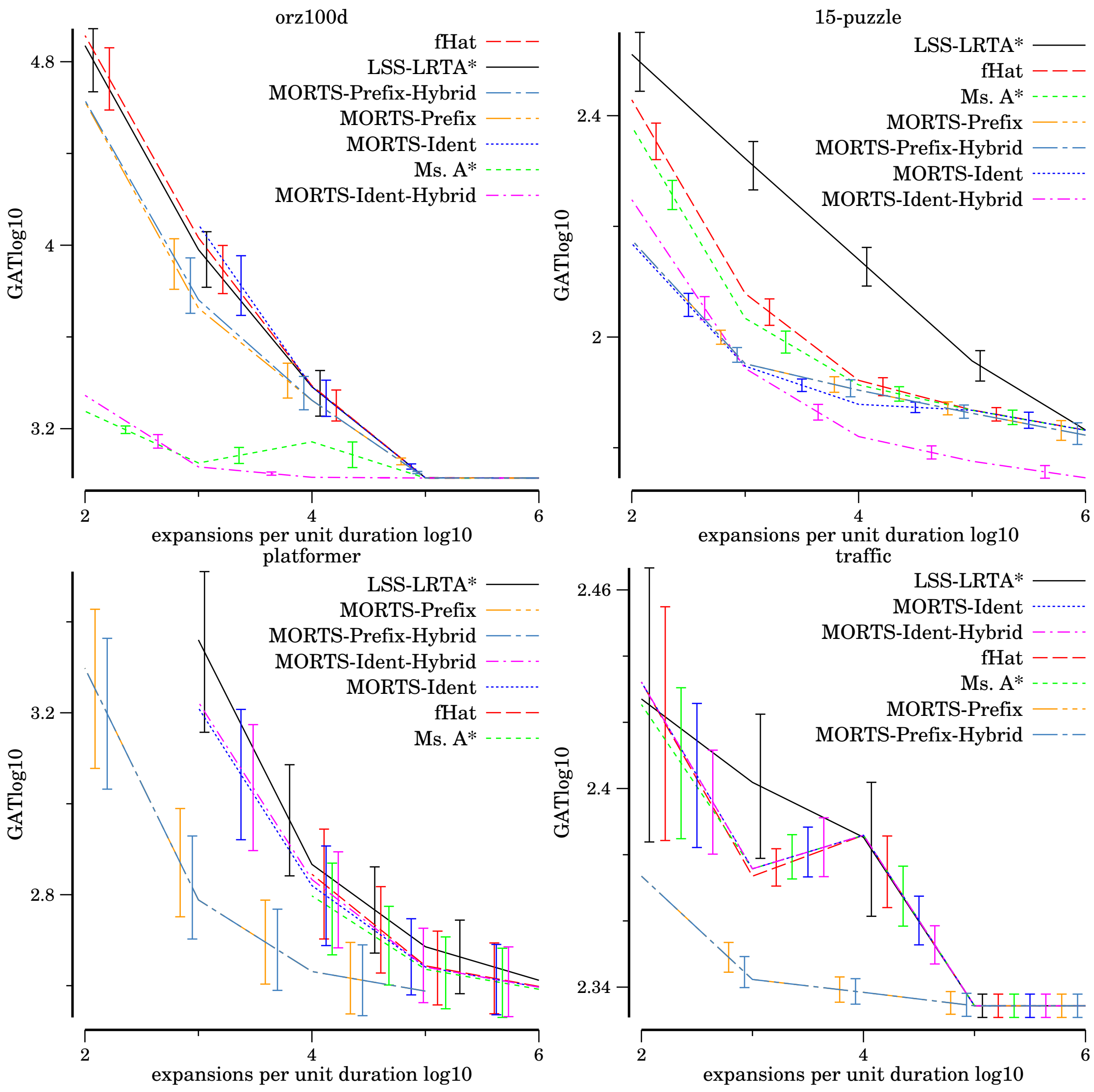


Figure 4-3: GAT performance of the Static MORTS-Prefix algorithm variants

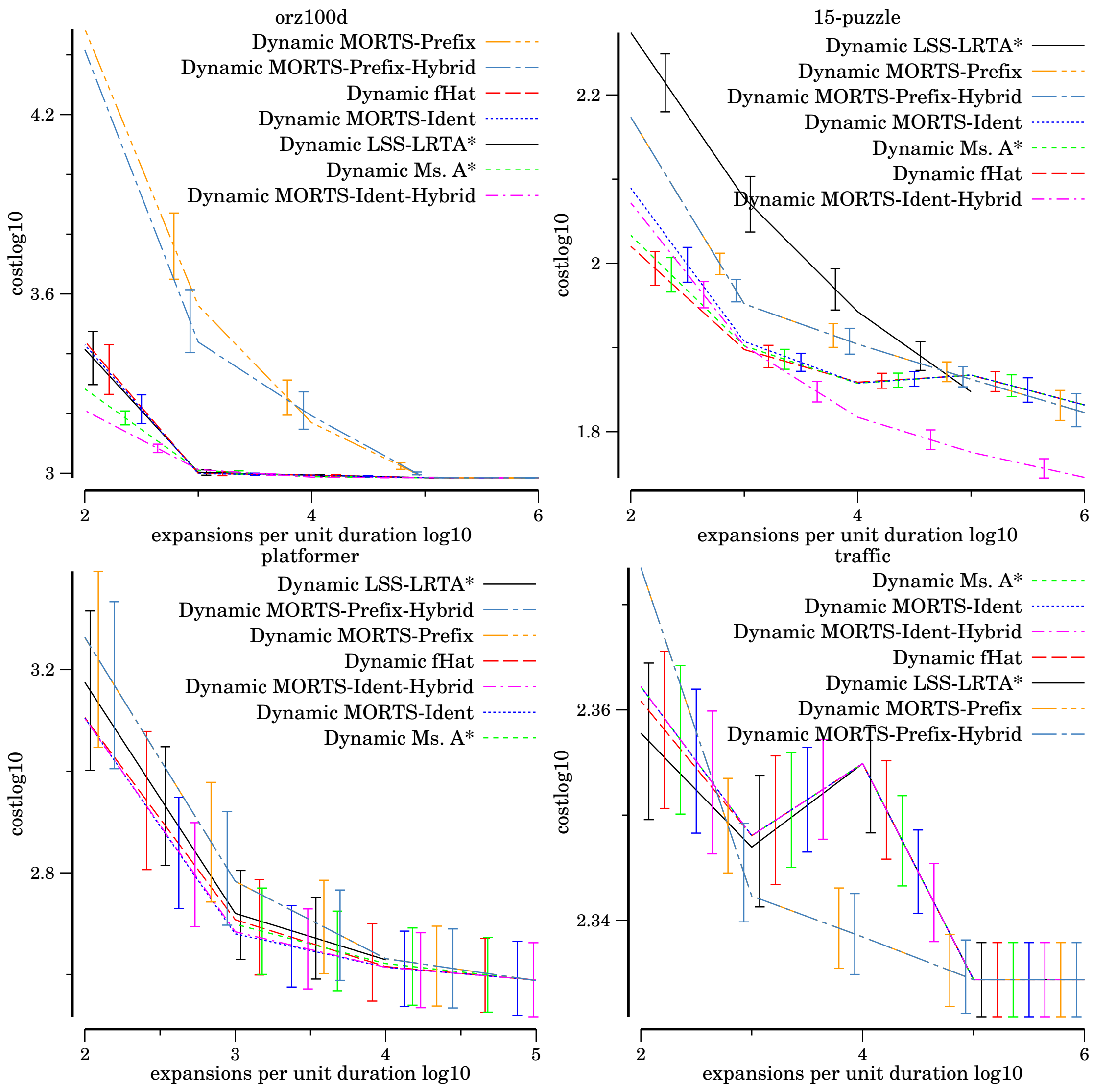


Figure 4-4: Total solution cost performance of the Dynamic MORTS-Prefix algorithm variants

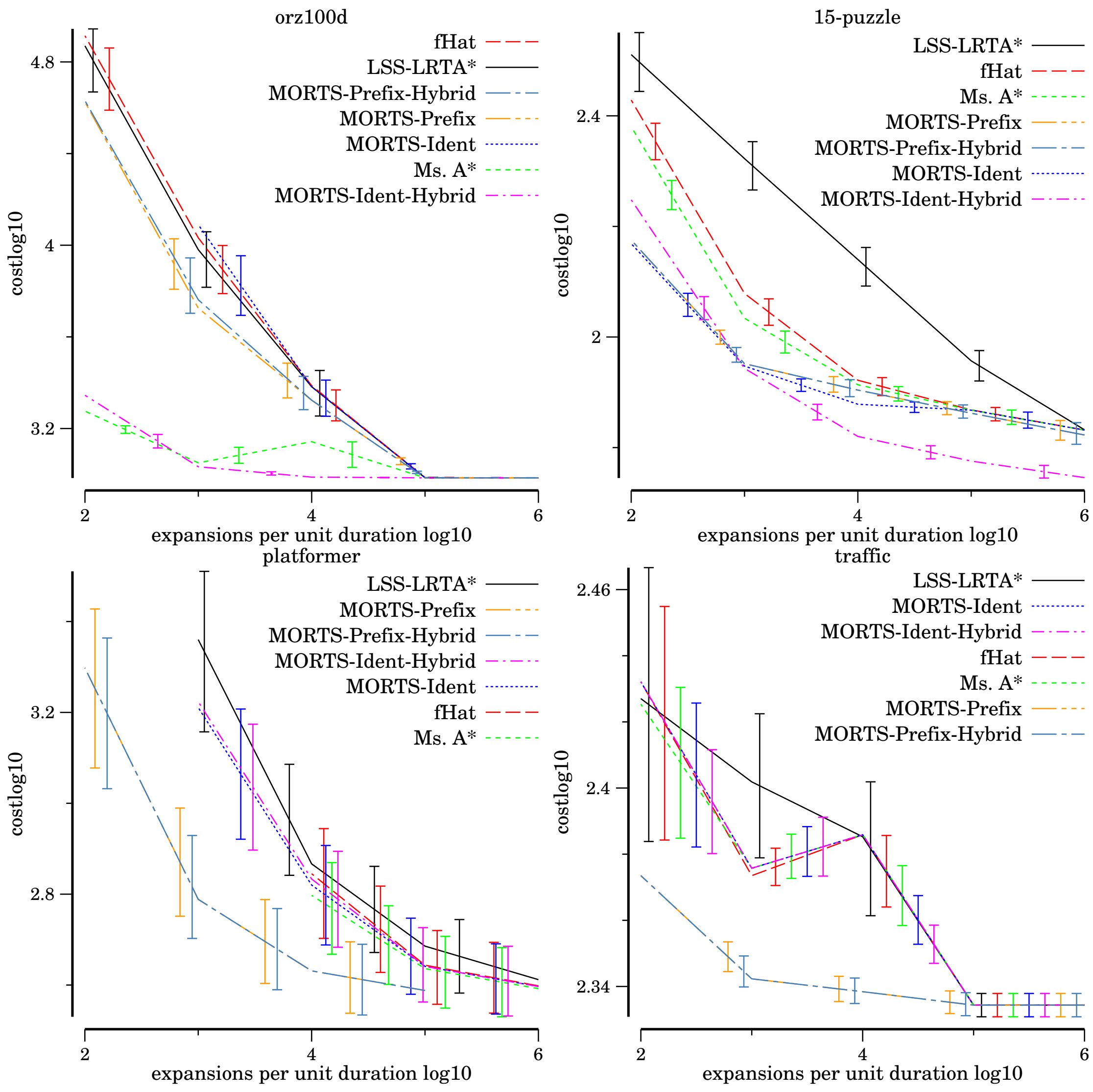


Figure 4-5: Total solution cost performance of the Static MORTS-Prefix algorithm variants

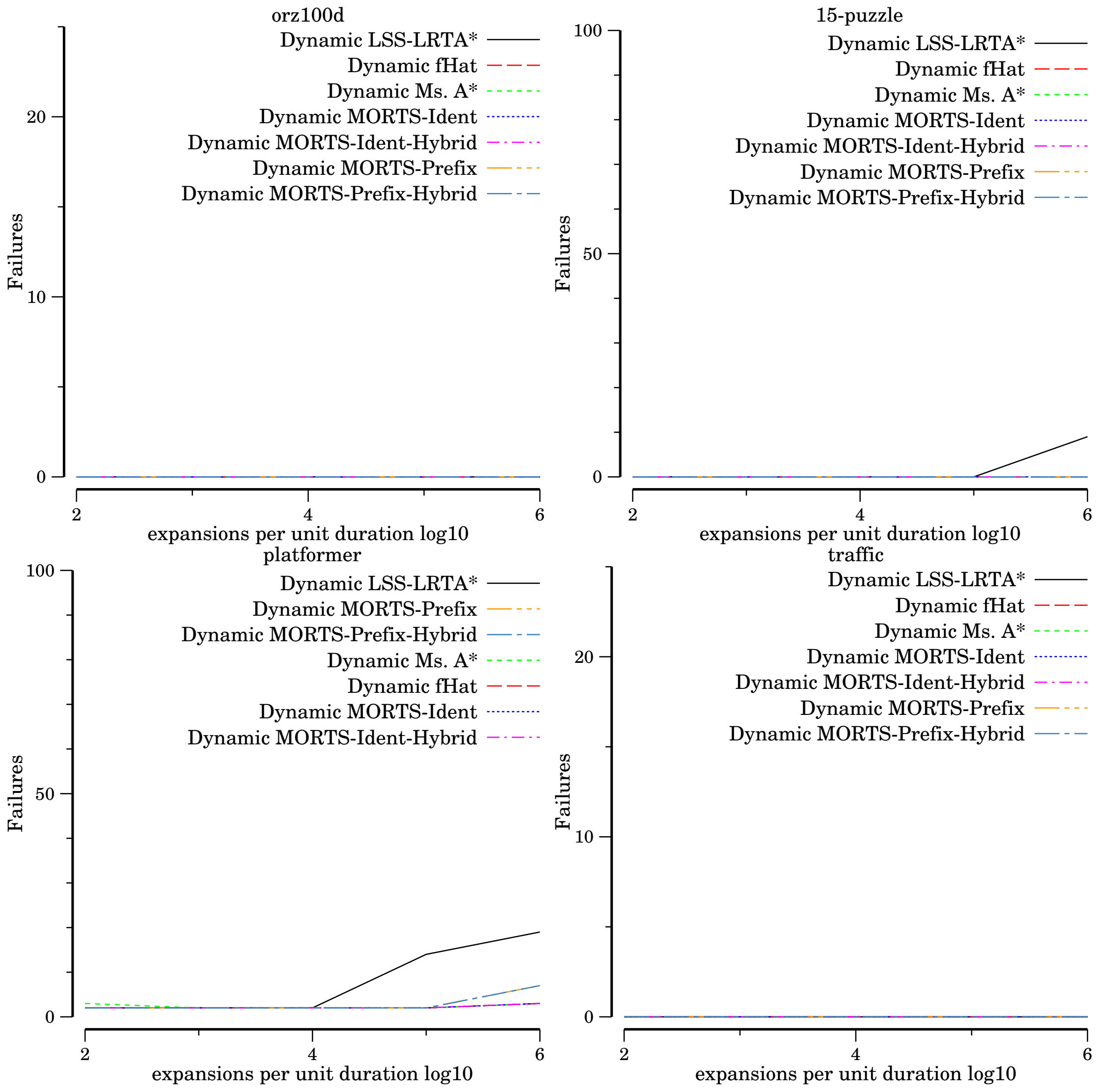


Figure 4-6: Failure rates of the Dynamic MORTS-Prefix algorithm variants

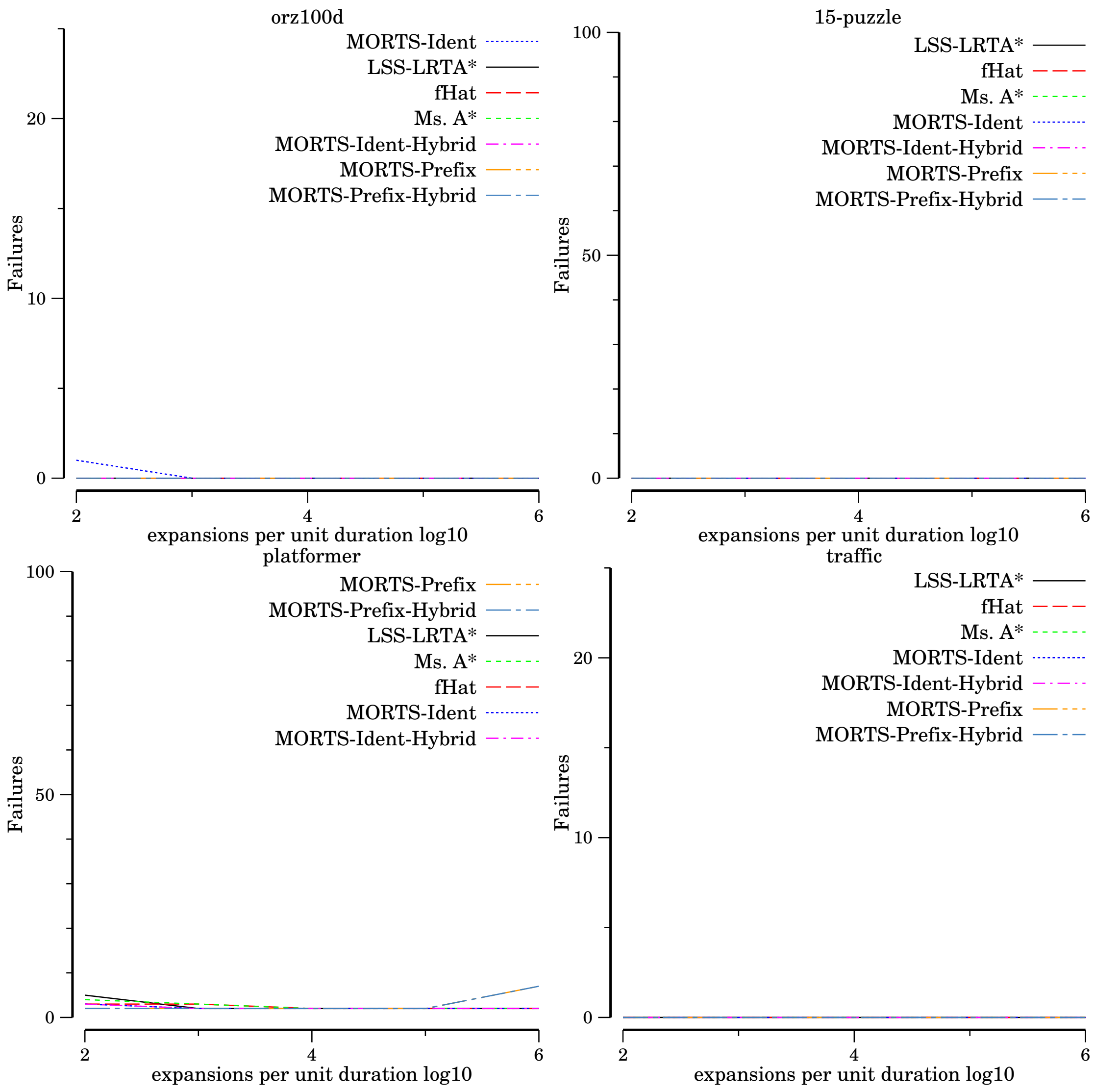


Figure 4-7: Failure rates of the Static MORTS-Prefix algorithm variants

lowing the Static lookahead paradigm, with some exceptions: in the Static paradigm, search is given the same amount of time for each iteration, regardless of the action or trajectory executed. As such, there is no loss of search time for subsequent iterations if the current path is cut short. In the Dynamic paradigm, however, a shorter queued path means less time for the next iteration of search, and though it is unclear how to quantize this loss, there is a distinct trade.

Despite the overall poor performance of the Dynamic MORTS-Prefix variant, it does outperform competitors on the traffic domain by a significant margin when they see a spike in GAT and solution cost. This is likely because Ms. A*'s meta level decision analysis is too weak to identify the junction at which the competing searches all fall into a dead end path, and because there are not identity actions available for MORTS-Ident to take. It should be noted that while the Dynamic lookahead paradigm typically produces the best performance for real-time searches, it may not always be appropriate. There may be, for instance, non-fully deterministic dynamic obstacles which invalidate queued actions before the agent can execute them. In such cases, there is an additional bound on the duration of search iterations which does not scale with the length of the queued path. When this occurs, it may be more desirable to exhibit behavior like MORTS-Prefix and cut the path short at an area of interest, as there is reduced or no loss of future search time.

Chapter 5

Custom Instances

In addition to the previous empirical evaluation which showed average performance of each algorithm over a number of instances for several domains, it is worth considering smaller instances which have been hand-crafted to provide insight into the behavior of a metareasoning search under various scenarios. For this purpose we present four instances of the grid pathfinding domain for which there is a distinct differential in the goal achievement times of a complete offline optimal search (A*) and greedy local hillclimbing (RTA* with a depth limit of 0).

5.1 Design

The instances used for this evaluation were designed such that for each there was a substantial difference in the GAT of A* and real-time hillclimbing implemented as RTA* with a depth limit of 0. The intuition behind this was that an effective metareasoning real-time search should exhibit behavior reminiscent of either of these searches when it is advantageous to do so, yielding good performance regardless of whether A* or hillclimbing performed best. Four instances in total were designed and can be seen in Figures 5-1, 5-2, 5-3, and 5-4.

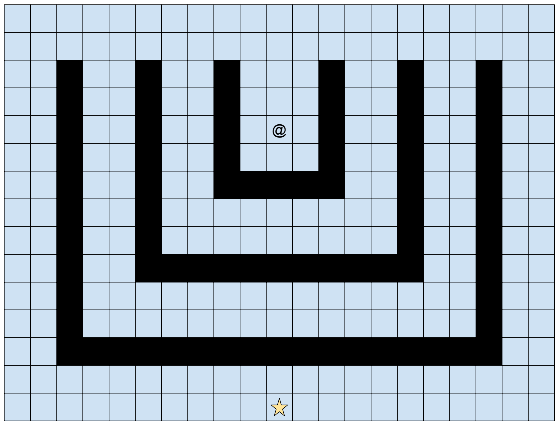


Figure 5-1: Custom Instance: Nested Cups

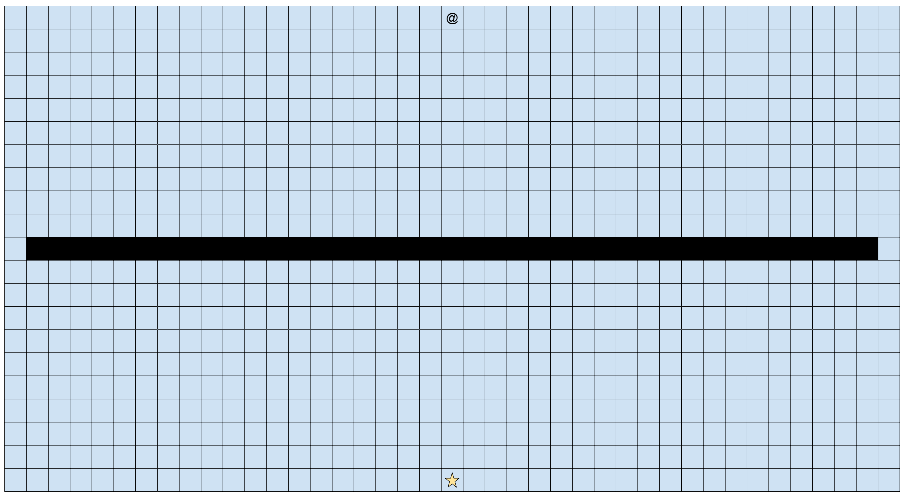


Figure 5-2: Custom Instance: Wall

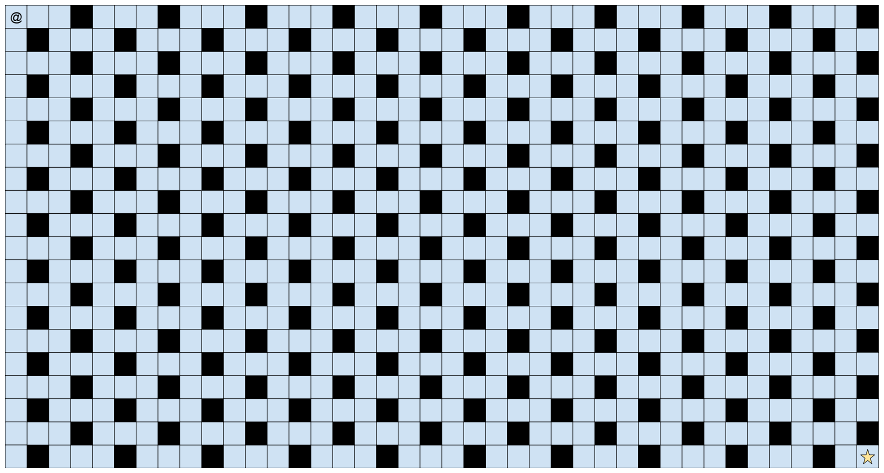


Figure 5-3: Custom Instance: Uniform

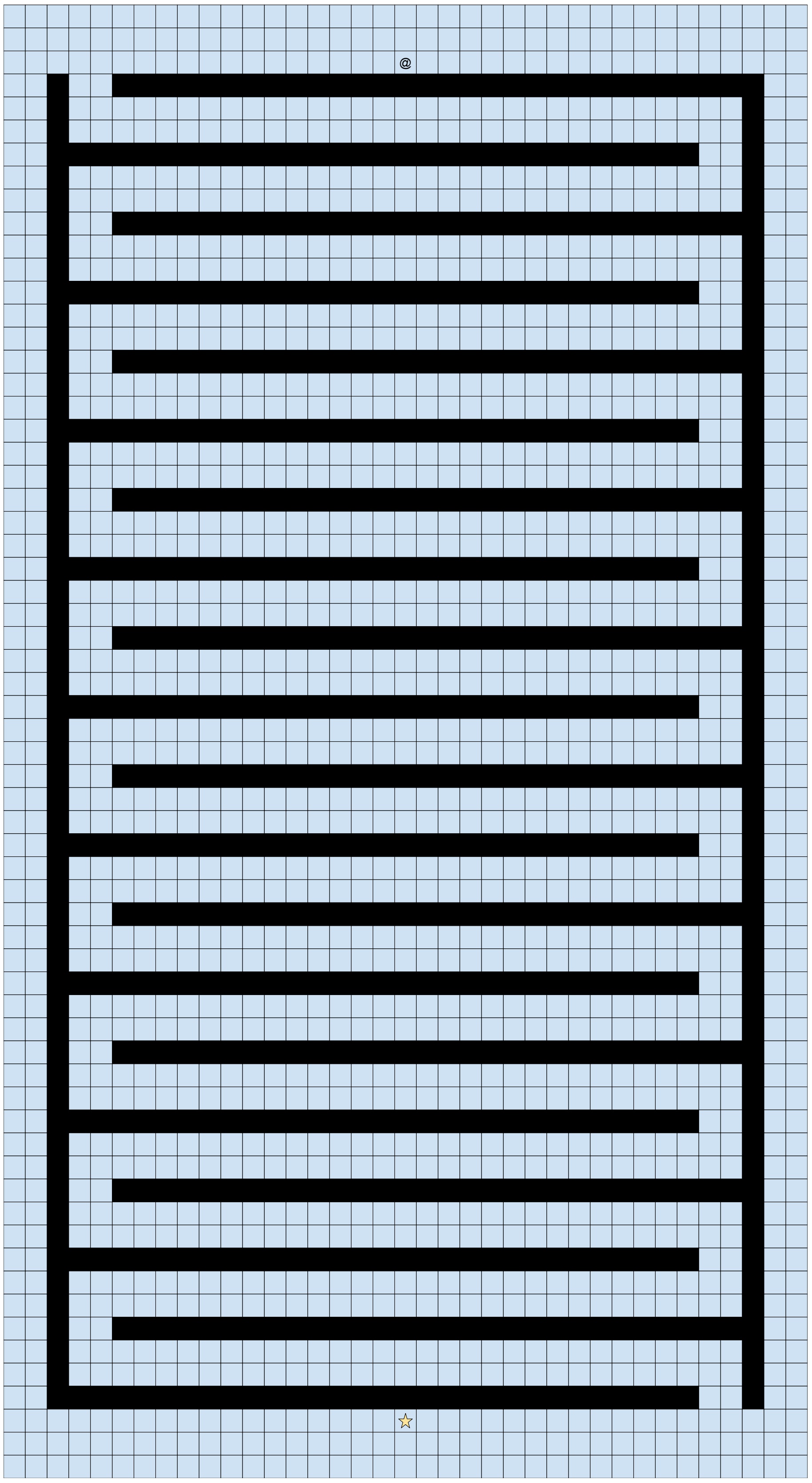


Figure 5-4: Custom Instance: Slalom

Algorithm	GAT	short trajectories	identity actions
A*	50.7	167	167
Hillclimbing	271.0	1	1
RTA*	229.0	1	1
LSS-LRTA*	255.0	1	1
$\hat{f}$	435.0	1	1
Ms. A*	125.0	13	11
MORTS-Ident	111.0	31	31
MORTS-Ident-Hybrid	73.0	21	21
MORTS-Prefix	159.0	104	1
MORTS-Prefix-Hybrid	159.0	104	1

Table 5.1: Performance in the Nested Cups instance

5.2 Evaluation

For the purpose of evaluation, we measured the GAT performance of each algorithm as well as the number of short trajectories and identity actions taken. A short trajectory denotes any time a search has the agent execute fewer actions than it has queued along its perceived best path. For any algorithms other than Ms. A* and MORTS-Prefix-Hybrid, this is limited to an identity action. An identity action is recorded for any action duration during which the agent is not moving. For all algorithms, the first search iteration is considered an identity action. For A*, all search iterations are identity actions as the agent does not move until search is completed.

The Nested Cups instance for which the results are shown in Table 5.1 is designed such that all obstacles create increasingly large areas of heuristic local minima. One would expect A* to outperform Hillclimbing in this case as once a state has been

Algorithm	GAT	short trajectories	identity actions
A*	102.1	421	421
Hillclimbing	241.0	1	1
RTA*	723.0	1	1
LSS-LRTA*	523.0	1	1
$\hat{f}$	717.0	1	1
Ms. A*	140.0	32	32
MORTS-Ident	493.0	111	111
MORTS-Ident-Hybrid	155.0	71	71
MORTS-Prefix	181.0	146	1
MORTS-Prefix-Hybrid	181.0	146	1

Table 5.2: Performance in the Wall instance

expanded it will not be revisited, while Hillclimbing and similar real-time searches revisit the same states repeatedly as they build up enough learning to escape the trap. One might also expect $\hat{f}$ to perform relatively well here, as with enough time in the inner cups, the error model leads it to trust the heuristic less. However, this is not the case, as it seems that the error model in fact prevents it from following the outside of the instance to the goal more strongly than with the standard heuristic model used by LSS-LRTA*. Each of the metareasoning algorithms performs well, however, suggesting that the certainty evaluations lead search to avoid local minima more effectively than relying on $\hat{f}$ to push the agent out of them.

The results of the Wall instance in Table 5.2 may appear unintuitive at first glance. One might expect the agent of hillclimbing or real-time search to simply follow the wall in either direction after hitting it until it reaches the end and continues unobstructed to the goal. In either direction along the wall, however, the heuristic estimate for

Algorithm	GAT	short trajectories	identity actions
A*	91.8	338	338
Hillclimbing	78.0	1	1
RTA*	77.0	1	1
LSS-LRTA*	81.0	1	1
$\hat{f}$	77.0	1	1
Ms. A*	77.0	1	1
MORTS-Ident	81.0	5	5
MORTS-Ident-Hybrid	84.0	8	8
MORTS-Prefix	75.0	58	1
MORTS-Prefix-Hybrid	75.0	58	1

Table 5.3: Performance in the Uniform instance

remaining cost-to-go is steadily increasing, and so search directs the agent back the way it came. The reason that Hillclimbing handles this situation better than RTA* is likely because of the mass of inflated heuristics in the center after the agent has wandered around. For Hillclimbing this mass is consistently generated from one neighbor to the next, whereas RTA* has heuristic inflation taken from all over its local vicinity depending on the current shape of the mass. As a result, Hillclimbing is slowly but consistently pushed along the horizontal axis in one direction, vacillating vertically, while RTA* vacillates both horizontally and vertically. Once again $\hat{f}$ fails to efficiently push the agent out of the local minima. The global error model also does not effectively guide MORTS-Ident. The path-based model seems to handle the situation better, however, likely because path based error grows consistently along the wall, while the global model does not. The prefix variants also handle the obstacle well, behaving like RTA* with less horizontal vacillation.

Algorithm	GAT	short trajectories	identity actions
A*	111.2	202	202
Hillclimbing	994.0	1	1
RTA*	1087.0	1	1
LSS-LRTA*	322.0	1	1
$\hat{f}$	578.0	1	1
Ms. A*	357.0	23	16
MORTS-Ident	111.0	10	10
MORTS-Ident-Hybrid	108.0	13	13
MORTS-Prefix	1332.0	790	1
MORTS-Prefix-Hybrid	1332.0	790	1

Table 5.4: Performance in the Slalom instance

The Uniform instance, the results for which are in Table 5.3, was designed with the expectation that Hillclimbing would outperform A*, as was shown to be the case. This is because the obstacles do not create any local minima even though there is obstacle obstruction. Consequently, real-time search can more or less follow the heuristic directly to the goal. It is interesting to note that, while not a substantial difference in performance relative to the other real-time searches, both variants of MORTS-Ident choose to take a number of identity actions due to uncertainty caused by the obstacles, despite the relative reliability of the heuristic.

The performance results for the Slalom instance in Table 5.4 show promise for MORTS-Ident. While other real-time searches naïvely followed an overly optimistic heuristic down the long and winding path, both variants of MORTS-Ident successfully detected the heuristic error and searched for a better alternative, leading the agent down the short path along the side. Problems such as this can pose significant difficulty for real-time searches, as evidenced by the performance of competing algorithms. It is not entirely clear why LSS-LRTA* was able to escape the trap faster than $\hat{f}$. The simplest explanation is an antagonistic search order (as search ordered on $\hat{f}$ is slightly less predictable than on f) causes learned values to propagate in a way that misleads the agent further as it tries to escape.

Chapter 6

Metareasoning Issues in Real-Time Search

This thesis has, up to this point, addressed the concept of using normal distributions on the belief of future heuristic estimates to compute the utility of search centered around some state. We have evaluated against competitor algorithms in both generic and handcrafted problem instances, and have discussed performance and insights for each. Despite this, there are many questions left unanswered with regards to metareasoning in real-time search that fall outside the scope of this work and remain to be addressed. In this chapter, we present a compilation of many such problems encountered in researching this topic.

6.1 Value of Identity Actions

When given some consideration, it seems intuitive that there is some inherent positive value in states that have identity actions available, though it is currently unclear how to quantify this value.

Consider, for instance the following scenario: the agent sits at the top of some large structure with the goal of reaching the bottom safely and quickly. There are three actions immediately available to the agent: an identity action, and a slide on either side. After a single iteration of search, the agent learns that the slide to its left will send it plummeting to certain doom, while the outcome of the right slide is less certain, and in fact this slide forks out not too far down (assume one unit action duration) into two separate paths. Unless the agent chooses to remain in its current predicament, there are two viable options remaining; either take the right path and hope to have an informed decision by the time it reaches the fork, or take the identity action in order to learn not about the immediately available actions but rather the options stemming further down the path to the right. Now consider that this initial state is not actually the top of the tower. Instead, the agent starts at the top of the tower already in motion, heading down a slide which forks in two directions. On the left is the state previously proposed as the top of the tower. On the right is a nearly identical state (with corresponding available actions and successor states), but without an identity action available. Assume that the agent has time before the initial fork to search further down each path, but not enough to form a complete path to the goal. From what the agent has learned, the two sides of the fork are identical, with the exception of the identity action on the left. Naturally, a rational agent should fork to the left because of an inherent value in the identity action it provides, however because this value is difficult to quantify, current algorithms do not consider it.

6.2 Inheritance of Identity actions

Often during search an agent may encounter a state s with an identity action but not enough uncertainty between α_s and β_s to warrant taking it. It may be, however, that somewhere along the agent's queued trajectory is a state s' for which there is enough uncertainty to warrant additional search but not enough actions from s to s' to sufficiently expel the uncertainty. If the uncertainty under s' is substantial enough to warrant an identity action, it makes sense that the agent may commit to the path prefix from s to s', but first emit one or more identity actions at s with the search centered around s'. As such, any state s' under a state s with an identity action available can be said to have that identity action available until the agent moves through s. This problem is largely distinct from the issue of assigning value to states with identity actions, however there is some carryover in that states which inherit identity actions should then also inherit the associated value. It should be noted that this inheritance is history dependent and does not carry over to when such a state is revisited in later search iterations.

6.3 Overoptimism in real-time search

Throughout the course of this paper we have discussed and presented algorithms for real-time search that direct the agent along what is believed to be the optimal path to the goal. It has already been established, however, that an optimal cost solution is not likely to be achieved for most real-time search problems, as decisions are being made using incomplete information. It is unlikely, unless we are using an exceptionally accurate heuristic, that the cost-to-go estimate of any given state denotes the actual remaining cost of the path the agent will traverse after passing through that state. In the case of many heuristics, especially those that are oblivious to obstacles, the heuristic may be wildly inaccurate in such a way that existing online correction techniques cannot effectively resolve. If the agent follows the heuristic estimate down a path of such states before search reveals the true future cost, it should have some agreeable alternative to which it can default, or else it is forced to, at best, retrace its steps until such an alternative avails itself.

It is yet unclear how to quantify the benefit of having multiple alternatives with similar expected total solution costs. It does seem important, though, to show some favor to states and paths for which there are a number of promising alternatives should the cost-to-go of the first choice prove higher than expected. One possible approach to account for this problem is to only commit to (or otherwise favor) states for which there are sufficiently many descendants whose total solution cost estimates lie within some ϵ of the best option. Alternatively, a search may choose some arbitrary number of leaves from the frontier, or the set of leaves whose total solution cost estimates precede some threshold, and select a path prefix such that the majority of the selected leaves are descendants of the terminal state.

6.4 Terminal Trajectory Commitment

In many real-time searches, including both those listed in Previous Algorithms and the newly presented metareasoning approaches, there is an explicit decision that once a goal state has been generated, the agent is to follow the found trajectory to that state and terminate search. In the case of optimal algorithms such as A*, this is a logical course of action, as the found path cannot be improved upon. In the case of real-time search, however, it is entirely possible that the goal can still be reached sooner if additional search is employed. As such, if there is any computation time left available for free, as is expected to be the case if the agent is still emitting a queued action, then it is intuitively incorrect to commit to a sub-optimal terminal path. It is difficult to say what exactly the best course of action is as to when or where to search once a terminal trajectory has been generated, but it is clearly the case that something should be done as long as computation time is available and the remaining solution is not provably optimal.

6.5 Early Termination of Search

One of the biggest advantages of path and path prefix commitment over single step commitment is that, when following the dynamic lookahead model of real-time search, longer path commitments allow for more search in the subsequent iteration. It may be, however, that before reaching the end of the path to which the agent has committed, the search may have enough certainty about the search space centered around the terminal state to commit to additional actions. If this is the case it may be worth terminating the current iteration of search early, extending the current path, and centering search around the new terminal state. This allows search to dedicate its efforts to areas of less certainty. Because the current path is deemed sufficiently certain, it is safe to perform the backup learning process on the current local search space and start a new one at the selected state from which we can expect to cover more new states of interest faster as there are fewer states competing on the frontier, all of which begin around the state which we already deemed best. Because this process involves committing to a state with some uncertainty while we still have search time freely available, it is unclear how to evaluate the utility of such behavior. One simple approach could be to simply extend the path prefix meta level decision to the current search space, but over commitment might lead search to waste resources, following an ultimately suboptimal path without exhausting the accumulated extra search time.

6.6 Belief Distribution Accuracy

One of the biggest problems facing the design of the presented metareasoning algorithms is formation of a belief distribution that accurately covers the relative likelihood of f^* having any given value. In the design presented in this paper, we make a relatively simplistic assumption that f^* lies along a normal distribution around some value; however it has been shown that the means of these distributions do not generally match the means of the corresponding f^* values that they attempt to estimate. It is also unclear how the f^* values themselves are not distributed relative to available estimates, further obfuscating the appropriate course of action. A simple but possibly suboptimal approach to solving this problem may be to take the collected sample data and use it to compute some function as an appropriate estimate of f^* given the available input.

6.7 Future Delay Estimation

We discussed earlier the practice of using expansion delay of the most recent search iteration to estimate how many steps toward the goal the next iteration of search will progress. This has shown to be relatively effective in practice, but has some problems theoretically and may not be the best approach. For instance, the expansion delay of the most recent iteration is expected to be an under estimate of future expansion delay, as the larger the search frontier becomes, the more states will be competing for expansion. This means that the next iteration of search will most likely not expand the frontier toward the goal as much as we have estimated. Additionally, as α and β vacillate, so does the expansion delay of each, with whichever state currently denoted as α being by definition the next to be expanded, often to have its best successor take its place and immediately be expanded. This means that as the means of the distributions change, so does the rate at which their variances change. It is not clear how to solve the latter of these two problems, but the former may benefit from extrapolation and interpolation of future expansion delay based on the average expansion delay relative to search space depth as seen previously during search.

6.8 State Loops

As we have seen from the traffic domain, there are scenarios in which identity actions are not available or otherwise not feasible. In some such cases, however, it is possible for an agent to emit a series of actions which, while not individually identity actions, lead the agent back to a previous state. We refer to such a series of actions as a state loop. A state loop serves the same function as an identity action, with its duration determined by the size of the loop. One example of a state loop would be an out-of-control car turning in circles while it figures out what to do next. The agent in this case lacks the ability to not change its state, but can follow a path that is guaranteed to return it to its current state. Another example is in the traffic domain. Every path from the start state that results in a collision is actually a state loop, as the instance is returned to the start state upon collision. One obvious means with which to detect a state loop is to run a cycle detection algorithm on the local search space at the end of a given search iteration.

6.9 Partial Expansion

In addition to his work on DTA*, Stuart Russel presented a method for evaluating the benefit of generating individual successors of a state [13]. By employing metareasoning practices in such a fine-grained manner, the search space may in theory be pruned significantly, thus allowing the local search space to expand deeper than it otherwise might. Coupled with the Dynamic lookahead paradigm and the possibility of identity actions, queued paths could be extended much further while maintaining some quantitative certainty about the correctness of the path. Additional information might also be used to influence the pruning of state generations in the case of real-time search, where many states are likely to be ones already expanded in previous search iterations. This area of research could easily yield many avenues for future improvements.

6.10 Search ordering

In addition to the question of when to search and where to root it, the order in which nodes are expanded can have a tremendous impact on the performance of search. We have discussed the importance of making search decisions using skepticism and attention to the expected probability of an outcome. In addition to the algorithms presented in this thesis, some focus was spent on developing a search algorithm that would order search by the expected contribution to the certainty of the state with the best total solution cost estimate. The idea behind this algorithm is that if the certainty of the best top level action can be improved sufficiently by the end of search, following top level actions selected in this manner would intuitively be less prone to misleading the agent.

In order to accomplish this task, the search frontier would be segmented into separate queues under each top level action. The order of search would then be determined by the reduction in benefit of search from expanding each top level action. To compute this, we would first take the the benefit $\mathcal{B}_x^\alpha$ of searching from the respective depths of each top level action's best leaf all the way to the goal for every competing top level action x. We would then estimate the future belief distribution of x as a function of two possible outcomes. The first being that the leaf under x generated the new best leaf with the same total solution cost estimate and one step closer to the goal. The second being that no such state would be generated, meaning that the new belief distribution would be based on the second best leaf. Once we had the estimated future belief distribution, we would compute the new benefit of search. The difference between the original benefit of search and the new benefit of search would be the reduction of benefit of search.

The intuition behind this process is that search becomes less beneficial as the certainty of α increases. As such, if we could minimize the benefit of additional search, it would imply that whichever action currently denoted as α would be intuitively trustworthy. Unfortunately, this algorithm was never refined enough to warrant a more in depth discussion alongside the other presented algorithms, however the idea shows promise for future development.

Chapter 7

Conclusion

In this thesis we discussed the concepts of search, heuristic search, real-time heuristic search, metareasoning, and goal achievement time. We presented, evaluated, and discussed previous algorithms thoroughly and skeptically so as to highlight search behavior worth considering as well as areas for potential improvement. We presented new algorithms in the hopes of improving goal achievement time by using metareasoning in real-time heuristic search and once again evaluated and discussed their performance in detail, indicating both when and why these approaches succeed and where there is still room for improvement. Finally, we presented numerous problems that have come to light in researching these prior steps which were not within the scope of this paper, so as to provide a platform upon which future research can be built.

Of the ideas presented in this thesis, some bear reiterating. It is known that real-time search cannot be guaranteed, nor should it be expected, to return an optimal solution. It is also known that relative to comparable offline searches, real-time search has the advantage of better average performance relative to goal achievement time. For these reasons, it is necessary for any rational real-time search to behave in such a way that optimizes the value of the expected outcome rather than the optimistic value of the best possible one. As we have shown, there are numerous avenues with which to approach this problem, ready and waiting for future research.

Bibliography

[1] Hart, P., Nilsson, N., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Science and Cybernetics*, SSC-4(2):100-107, 1968.

[2] Hernandez, C., Baier, J. Urtas, T., and Koenig, S. (2012). Time-bounded adaptive A*. In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-12)*

[3] Ethan Burns, Wheeler Ruml, and Minh Binh Do. Heuristic search when time matters. *J. Artif. Intell. Res. (JAIR)*, 47:697-740, 2013.

[4] Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games, 4*(2), 144-148.

[5] Korf, R. E. (1987) Real-time heuristic search: First results. *AAAI*, 90-94.

[6] Korf, R. E. (1988) Real-time heuristic search: New results. *AAAI*, 139-144.

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

[8] Koenig, S., and Sun, X. (2009). Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems, 18*(3), 313-341.

[9] Russell, S., and Wefald, E. (1991). *Do the right thing: studies in limited rationality.* The MIT Press.

[10] Kaelbling, L. P. (1993). *Learning in embedded systems.* The MIT Press.

[11] Thayer, J. T., Dionne, A., and Ruml, W. (2011). Learning inadmissible heuristics during search. In *Proceedings of the Twenty-first International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 333-340. International Foundation for Autonomous Agents and Multiagent Systems.

[12] Dionne, A. J., Thayer, J. T., and Ruml, W. (2011). Deadline-aware search using on-line measures of behavior. In *SOCS*.

[13] Russell, S. (1990). Fine-grained decision-theoretic search control. In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence (UAI).*