Winter 2021

Metareasoning for Heuristic Search Using Uncertainty

Tianyi Gu
University of New Hampshire

METAREASONING FOR HEURISTIC SEARCH USING UNCERTAINTY

BY

Tianyi Gu

MEng of Logistics Engineering, Shanghai Maritime University,
Shanghai, China 2012
BEng of Logistics Engineering, Shanghai Maritime University,
Shanghai, China 2010

DISSERTATION

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Doctor of Philosophy

in

Computer Science

December, 2021

This dissertation has been examined and approved in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science by:

Dissertation Director, Wheeler Ruml
Professor of Computer Science,
University of New Hampshire

Momotaz Begum
Assistant Professor of Computer Science,
University of New Hampshire

Laura Dietz
Assistant Professor of Computer Science,
University of New Hampshire

Levi Lelis
Assistant Professor of Computer Science,
University of Alberta

Marek Petrik
Assistant Professor of Computer Science,
University of New Hampshire

On November 17th, 2021

Approval signatures are on file with the University of New Hampshire Graduate School.

To my wife and family.

ACKNOWLEDGMENTS

This long journey far from home would not have been possible without the endless and unconditional support of a stream of people who have carried me forward over the years and prevented me from giving up.

First, I would like to thank my supportive family, especially my wife, Lingling. The courage she gave me guided me through the most challenging times during my PhD. She understood how much I enjoy what I do. She helped me take care of most chores, ensured we lived a decent life even with very limited resources and gave me enough time to finish this part of my life.

I'm forever grateful for the mentorship of Wheeler Ruml. I have learned more from Wheeler than I ever anticipated. He was always available whenever I needed guidance and could always provide unique perspectives to move things forward. He taught me to move toward excellence and showed me many ways to reach that. He encouraged me to be the unusual and extraordinary guy since no great achievement went along with the crowd.

This work would have been impossible without the long discussions, numerous comments, and technical help from Bence, Max, Leo, Shahaf, Eyal, Erez, Jörg, Marek, Reazul, Bahram, Andrew, Yi, Steve, Devin, and many others. I also would like to thank all the members of my dissertation committee for providing me with insightful feedback and comments.

I could not have made it this far without the support and care of my friends outside of my academic world, including Yunyao, Aiting, Chang, Monkie, Kevin, Mostafa, and many others. They made my six and half years in grad school more survivable, more colorful, and more fulfilling. Many wonderful new friends from Dover Adult Learning Center, especially our teachers, Bill and Rosemary, shaped my view of the world and taught me a surprisingly tremendous amount about life in general.

Lastly, I gratefully acknowledge support from the NSF-BSF (grant 2008594), the UNH Department of Computer Science, and the UNH Graduate School.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGMENTS iv

LIST OF TABLES vii

LIST OF FIGURES x

ABSTRACT xi

Chapter 1 Introduction **1**

1.1 Metareasoning for Real-time Heuristic Search 3

1.2 Metareasoning for Bounded-cost Search 4

1.3 Metareasoning for Situated Planning 5

1.4 State Space Search 5

1.5 Conclusion 7

Chapter 2 Metareasoning for Real-time Heuristic Search **8**

2.1 Introduction 9

2.2 Background 12

2.3 Data-Driven Nancy 20

2.4 Other Distributional Methods 43

2.5 Conclusion 49

Chapter 3 Metareasoning for Bounded-Cost Search **50**

3.1 Introduction 50

3.2 Previous Work 52

3.3 Expected Effort Search 54

3.4 BEES with Explicit Probability Estimates 57

3.5 Experimental Evaluation 58

3.6 Conclusions 74

Chapter 4 Metareasoning for Situated Planning **75**

4.1 Introduction 75

4.2 Background 76

4.3 Metareasoning for Action Commitment 80

4.4 Empirical Evaluation 86

4.5 Conclusion 89

Chapter 5 Conclusion **92**

Bibliography **94**

LIST OF TABLES

2-1 Geometric means of the solution cost on instances solved by all algorithms. The limit on the number of expanded nodes in the lookahead is denoted by L. 37

3-1 Coverage on the IPC'18 bounded-cost instances, and geometric means of the expansions (multiply by 10^3) and search time on commonly solved instances (last two rows). 71

3-2 Coverage on the IPC satisficing instances. The normalized overall coverage corrects for the different numbers of instances per domain. The last two rows show the geometric means of the expansions and search time on commonly solved instances. 73

LIST OF FIGURES

1-1 Illustration of optimal action selection with time cost (Figure 1.2 from (Russell & Wefald, 1991)). 2

2-1 Should an agent at A move to B_1 or B_2? (Pemberton & Korf, 1994) 10

2-2 Should an agent expand nodes under α or β? 11

2-3 Iterations of LSS-LRTA*. 13

2-4 Beliefs generated for uniform-cost 15-puzzle. 22

2-5 Beliefs generated for 16-pancake. 23

2-6 Beliefs generated for Barto Racetrack. 23

2-7 Beliefs gathered for the Blocksworld and Transport (unit-cost). 24

2-8 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the uniform-cost variants of the 15-puzzle, with error bars indicating 95% confidence intervals. 30

2-9 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the heavy-cost variants of the 15-puzzle, with error bars indicating 95% confidence intervals. 31

2-10 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 16 pancakes. 32

2-11 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 32 pancakes. 33

2-12 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 40 pancakes. 34

2-13 Barto (left) and uniform (right) Racetrack maps. 35

2-14 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the Barto map of the Racetrack domain. 36

2-15 Solution cost relative to LSS-LRTA* with varying lookahead bounds on the uniform map of the Racetrack domain. 38
2-16 Solution cost as a function of CPU time on unit-cost 15-puzzle. 39
2-17 Solution cost as a function of CPU time on heavy-cost 15-puzzle. 40
2-18 Solution cost as a function of CPU time on 16 pancake. 40
2-19 Solution cost as a function of CPU time on 32 pancake. 41
2-20 Solution cost as a function of CPU time on 40 pancake. 41
2-21 Solution cost as a function of CPU time on Barto map racetrack. 42
2-22 Solution cost as a function of CPU time in uniform map racetrack. 42
2-23 Comparison to IE and THTS on uniform-cost 15 puzzle. 44
2-24 Comparison to IE and THTS on heavy-cost 15 puzzle. 45
2-25 Comparison to IE and THTS on 16 pancake. 45
2-26 Comparison to IE and THTS on 32 pancake. 46
2-27 Comparison to IE and THTS on 40 pancake. 46
2-28 Comparison to IE and THTS on Barto map racetrack. 47
2-29 Comparison to IE and THTS on uniform map racetrack. 47

3-1 Estimating $p(n)$. 57
3-2 Performance measurements as a function of the cost bound on uniform tile. . . 59
3-3 Performance measurements as a function of the cost bound on heavy tile. . . . 60
3-4 Performance measurements as a function of the cost bound on square-root tile. . 61
3-5 Performance measurements as a function of the cost bound on inverse tile. . . . 62
3-6 Performance measurements as a function of the cost bound on uniform vacuum world. 63
3-7 Performance measurements as a function of the cost bound on heavy vacuum world. 64
3-8 Performance measurements as a function of the cost bound on regular pancake. 65
3-9 Performance measurements as a function of the cost bound on heavy pancake. . 66

3-10 Performance measurements as a function of the cost bound on hansen map racetrack with Dijkstra heuristic. 67
3-11 Performance measurements as a function of the cost bound on barto map racetrack with Dijkstra heuristic. 68
3-12 Performance measurements as a function of the cost bound on hansen map racetrack with euclidean heuristic. 69
3-13 Performance measurements as a function of the cost bound on barto map racetrack with euclidean heuristic. 70
3-14 Normalized coverage as the cost bound is increased. 74

4-1 Committing vs not committing. 82
4-2 Schematics of grid benchmarks with tar pits. 87
4-3 Goal achievement time as a function of search speed in grid pathfinding with tar pits near the start. 89
4-4 GAT with corridor and tar pit near the goal. 90
4-5 GAT with tar pits at both ends. 91

ABSTRACT

METAREASONING FOR HEURISTIC SEARCH USING UNCERTAINTY

by

Tianyi Gu

University of New Hampshire, December, 2021

Heuristic search methods are widely used in many real-world autonomous systems. Yet, people always want to solve search problems that are larger than time allows. To address these challenging problems, even suboptimally, a planning agent should be smart enough to intelligently allocate its computational resources, to think carefully about where in the state space it should spend time searching. For finding optimal solutions, we must examine every node that is not provably too expensive. In contrast, to find suboptimal solutions when under time pressure, we need to be very selective about which nodes to examine. In this dissertation, we will demonstrate that estimates of uncertainty, represented as belief distributions, can be used to drive search effectively. This type of algorithmic approach is known as metareasoning, which refers to reasoning about which reasoning to do. We will provide examples of improved algorithms for real-time search, bounded-cost search, and situated planning.

CHAPTER 1

Introduction

Heuristic search methods are widely used in many real-world autonomous systems. For example, search-based methods have been used in motion planning for self-driving cars (Ferguson, Howard, & Likhachev, 2008) and Mars rovers (Mudgal, Tovey, Greenberg, & Koenig, 2005), task planning for robots (Karpas & Magazzeni, 2020), scheduling systems for container terminals (Kim & Park, 2004), multi-agent path planning for Amazon warehouses (Li, Tinka, Kiesel, Durham, Kumar, & Koenig, 2021), power scheduling (Thiébaux, Coffrin, Hijazi, Slaney, et al., 2013), and protein design (Allouche, Barbe, de Givry, Katsirelos, Lebbah, Loudni, Ouali, Schiex, Simoncini, & Zytnicki, 2019). However, people always want to solve search problems that are larger than time allows. To solve the challenge problems, even suboptimally, a planning agent must be smart enough to intelligently allocate its computational resources, to think carefully about where in the state space it should spend time to search. For finding optimal solutions, we must examine every node that is not provably too expensive. In contrast, to find suboptimal solutions when under time pressure, we need to be very selective about which nodes to examine. Considering an autonomous vehicle driving in an urban area: when a kid rushes into the road and is in front of a fast-moving vehicle, a planner might only have 0.1s of thinking time before committing to the next action. The agent might better first think clearly about how to use the time to plan (or maybe not to plan at all) rather than blindly start the planning, which can lead to failure of finding a solution; consequently, this can result in unplanned behavior of the vehicle (i.e., an emergency stop is triggered, which is not going to rescue the situation due to the inertia).

This type of algorithmic approach was named metareasoning, which refers to reasoning about reasoning. An agent equipped with a metareasoning component would solve a meta-level reasoning problem in addition to the conventional object-level reasoning problem (Horvitz, 1990). These

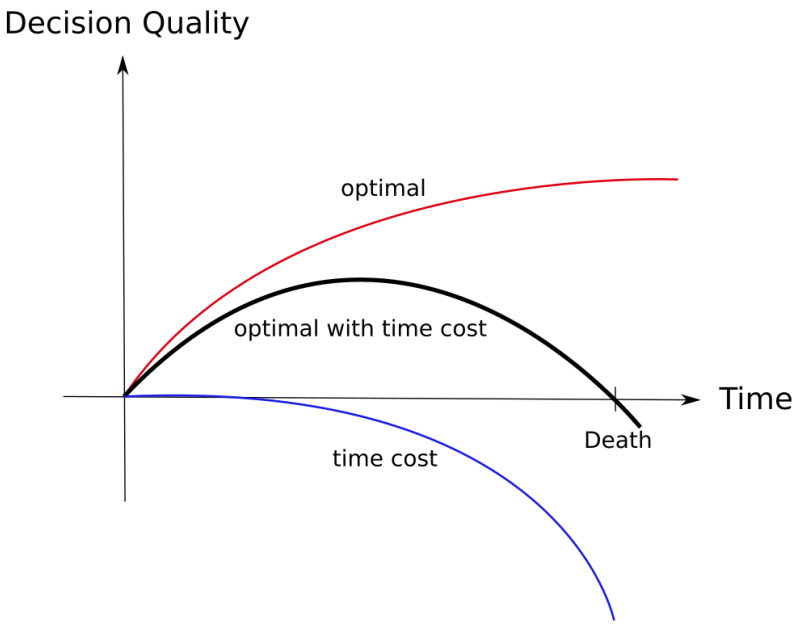


Figure 1-1: Illustration of optimal action selection with time cost (Figure 1.2 from (Russell & Wefald, 1991)).

two problems typically differ in their utility functions. The meta-level utility is the expected utility associated with inference-related cost (i.e., deliberation cost). In contrast, the object-level utility is the expected utility associated with the state of the world (i.e., solution cost) without regard to the cost of reasoning. In this dissertation, I propose practical metareasoning methods that carefully allocate effort due to time pressure and optimize combined utility that considers both reasoning cost and object-level cost.

Historically, the metareasoning problem has been discussed for a long time since it was proposed. Back in the 1960s, the statistician I.J. Good (Good, 1971) stated the conceptual distinction between "type I" rationality and a new "type II" rationality. The "type I" optimize the classical expected utility while "type II" instead maximizes the expected utility that also considers deliberation cost. This concept was then further investigated and formalized by Russell and Wefald, among others, in the late 1980s. In their famous book *Do the Right Thing* (Russell & Wefald, 1991), Russell and Wefald illustrated the optimization problem that consider the deliberation time using Figure 1-1. The red line shows the so-called 'intrinsic utility' obtained by the agent as time progresses. The solution quality would increase given more and more computational time. The blue line is the time

cost which would be getting worse and worse as time progress. The black line shows the 'net utility' of taking an action at a particular time, which is the intrinsic utility minus the time cost. They also proposed four meatareasoning architectures and formalized the problem. In 2000s, Zilberstein (Zilberstein, 2008) further discussed the computational models of bounded rationality and argued that the metareasoning component ought to offer desirable properties. Given so many elegant arguments and discussions about metareasoning in history, there have been not many practical algorithms that actually do metareasoning. For an agent to plan under time pressure, obviously, it should not only optimize the solution cost but also the planning time to achieve the best performance. Therefore, I pursue practical metareasoning approaches that can enhance various families of traditional search algorithms.

Conventional metareasoning approaches follow decision theory that tells the agent to select the action which maximizes the expected utility. However, when modeling an uncertain value, scalar expected value often is not as powerful as belief distribution. Intuitively, distribution helps because it quantifies uncertainty, which is what search resolves. Recently, distributional methods have been proposed in the RL community (Dearden, Friedman, & Russell, 1998; Bellemare, Dabney, & Munos, 2017). It has been shown that distributional-informed methods can often outperform the scalar-expected-value-informed methods by taking advantage of value uncertainty reasoning. Therefore, taking inspiration from this prior work from RL, this dissertation pursues an alternative class of rational metareasoning that takes advantage of distributional methods to have a better estimate model instead of only relying on the expected utility. The distributions can be constructed through offline learning or online learning.

The thesis of this dissertation is that heuristic search can benefit from representing uncertainty. In this dissertation, I will demonstrate the thesis by improving search algorithms for the following three problem settings: (1) real-time search, (2) bounded-cost search, and (3) situated planning.

1.1 Metareasoning for Real-time Heuristic Search

In chapter 2, I address real-time search, which is a practically relevant problem setting that is suitable for online planning (e.g., the autonomous vehicle example above) (Cserna, 2019). In this setting,

when the time-bound is reached, the agent has to commit to the best action on hand even if it only has a partial solution plan. Because the time-bound tightly limits the computation that the agent can perform, metareasoning can play an important role in this setting. Traditional real-time search methods were adapted directly from off-line search methods like A* (Hart, Nilsson, & Raphael, 1968) that make their online action selection based on a lower bound rather than an expected cost, and thus are not appropriate as a basis for rational action selection. To do a rational real-time search, it might be worth it for the agent to gather information about the value uncertainty due to the bounded rationality and make online decisions based on the expected utility and value uncertainty. The traditional real-time search approaches are lacking this kind of metareasoning component. Chapter 2 makes contributions to the design of a rational real-time search approach.

1.2 Metareasoning for Bounded-cost Search

In chapter 3, I propose algorithms for bounded-cost search settings where the agent is given a specific cost bound along with the search problem. The goal is to find a complete solution within a cost bound as quickly as possible. Bounded-cost search is also handy since its users can have control over the solution quality. Bounded-cost search is another suboptimal search setting in which carefully allocating search effort can greatly impact the overall performance. Thus our distributional metareasoning approaches can provide improvement against current state-of-art approaches. Traditional methods for bounded-cost search are focused on designing inadmissible heuristics that can guide the search toward the search nodes that have a high chance for finding a solution within bound (Stern, Puzis, & Felner, 2011). BEES (Thayer, Stern, Felner, & Ruml, 2012) explicitly tries to find a solution within the bound as quickly as possible, which is a meta-level question. However, the performance of BEES can be susceptible to the error of its estimate. In chapter 3 of my dissertation, I propose a distributional method to explicitly reason about this meta-level question and take advantage of knowing the estimate's uncertainty and thus better guide the search. This work further advances the trend of exploiting estimates of uncertainty, traditionally used in planning under uncertainty (Dearden et al., 1998; McMahan, Likhachev, & Gordon, 2005; Bellemare et al., 2017), in methods for deterministic single-agent search.

1.3 Metareasoning for Situated Planning

In chapter 4, I propose a metareasoning algorithm for the situated online planning problem setting, in which the search agent has to be aware of the planning time passing. Specifically, our method answers the question of when to commit to an action in online planning. When the planner commits to an action, it re-roots its search at the node representing that action's outcome. We assume that the system cannot be uncontrolled, so the planner must commit to a new action (perhaps a no-op) before the previously chosen action completes. In this setting, it can be beneficial to commit early to devote more lookahead search focused below an upcoming state. In the third part of my dissertation, I propose a principled method, called Flexible Action Commitment Search (FACS), for making this commitment decision. Fundamentally, we want to answer whether it is worth committing early so that the agent can allocate more search effort for a future state. Similar to real-time search, our search would encounter a lot of value uncertainty due to the online search setting, which provides a good testbed for investigating our distributional-metareasoning-based approaches.

1.4 State Space Search

In this section, we provide a brief introduction to state space search. In this dissertation, we rely on the following formulation of state space search to discuss heuristic search methods: Given a set of states S, a set of goal states G, a set of actions A, a successor function (Definition 1), a cost function (Definition 2), and a heuristic function h (Definition 3) and the objective is to find a sequence of actions such that applying those actions starting from the initial state lead to one of the goal states.

Definition 1 *The **successor function** returns a set of all possible direct successor states* s' *from a given source state* s *along with the action* a *that leads to the states* s' *from* s*:*

$$successor : s \in S, a \in A \rightarrow s' \subseteq S.$$

Definition 2 *The **cost function*** c *defines the cost of taking an action* a *from a given state* s *to a state* s'. *Due to the deterministic nature of the domains discussed in this dissertation this formulation is*

redundant and the following invocations of this function refer to the same value:

$$c(s, a, s') = c(s, a) = c(s, s').$$

Definition 3 *The* ***heuristic function*** h *estimates the cost of the lowest cost path between a state* s *to a goal state. This is also known as* s*'s heuristic value or simply the heuristic of* s*. The heuristic value of state* s *is denoted as* h(s)*.*

Definition 4 *A heuristic function* h *is* ***admissible*** *iff it never overestimates the true cost of reaching a goal state from any given node or more formally:*

$$admissible(h) \Leftrightarrow h(s) \leq h^*(s) : \forall s \in S$$

where $h^*(s)$ *is the true cost of reaching the closest goal from state* s*.*

Definition 5 *A heuristic function* h *is* ***consistent*** *iff the heuristic value* h(s) *of any state* s *is less than or equal to the heuristic value of any successor* s' *of* s *plus the cost of getting to* s' *from* s*:*

$$consistent(h) \Leftrightarrow h(s) \leq h(s') + c(s, s') : \forall s' \in successor(s).$$

This formulation implicitly defines a graph where the states are the vertices and the edges are given by the successor function. The size of this implicit graph is super-astronomical for many problems. Search algorithms discussed in the following chapters rely on an explicit tree (or graph) referred to as the search tree (or graph). The search tree is incrementally induced by the search algorithm. The search tree wraps the states (or set of states) from the state space with search nodes and connects these nodes with edges that may or may not exist in the original search space. The structure of the search tree is algorithm specific. The node is a simple data structure containing meta information recorded by the search method (e.g. updated heuristic value or parent node). While there is a clear distinction between states and nodes we will use them interchangeably if there exists a one-to-one mapping between them.

In the context of the search trees, we will refer to the invocation of the successor function on a state as expansion or node expansion. Expanding a node means that the successors of the state

represented by the node are added to the search tree as well as to the open list (see below) of the search algorithm, unless otherwise noted.

Throughout this work the algorithms used for state space exploration are based on best-first search. Best-first search algorithms expand nodes until a goal state is reached (expanded) starting from the initial state. The node to be expanded is selected from the successors of the previously expanded nodes based on a priority function that is defined by the algorithm. We refer to this set as open nodes and the data structure that stores it as open list.

Definition 6 *Given a search tree, the* g *value defines the cost of the path leading from the initial state to a node defined by the tree structure.*

Definition 7 *The* f *value of a node* n *is the sum of the* g *value of the node and the* h *value of the state* s *represented by* n.

$$f(n) = g(n) + h(s)$$

f is a commonly used function for best-first search algorithms (Hart et al., 1968).

The notation above of state space search is used throughout this dissertation. Additional functions (such as distance or d, $\hat{h}$, and $\hat{f}$) will be defined in the following chapters as needed.

1.5 Conclusion

This work hopes to encourage further efforts in widening the applicability of metareasoning for suboptimal search and planning under time pressure. The results in this dissertation demonstrate that heuristic search can benefit from rational decision-making through representing uncertainty. Metareasoning can be very powerful when an agent has to deal with complex problems, especially when resources are tight. Many kinds of suboptimal search and planning methods can benefit from adding a metareasoning enhancement, as illustrated by my work in real-time search, bounded-cost search, and situated planning.

CHAPTER 2

Metareasoning for Real-time Heuristic Search

In real-time planning, an agent must select the next action to take within a fixed time bound. Even in deterministic domains, action selection inherently suffers from uncertainty about those portions of the state space that have not yet been computed by the lookahead search. In this chapter, we explore how an agent can benefit from metareasoning about this uncertainty. We make the uncertainty explicit by treating heuristic values as uncertain evidence, inducing beliefs over the true remaining cost. Given this distributional perspective, a recent proposed real-time framework (Mitchell, Ruml, Spaniol, Hoffmann, & Petrik, 2019), called Nancy, re-considers the major components of real-time heuristic search, examining several alternative methods for lookahead strategy and value back-up. Nancy expands nodes to minimize the expected regret in case a non-optimal action is chosen. In our work, we show how Nancy's beliefs can be learned beforehand from training data; we call the new method Data-Driven Nancy (DDNancy). We compare Nancy and DDNancy experimentally against both conventional real-time search algorithms like LSS-LRTA* and approaches that exploit uncertainty, such as Monte Carlo tree search and Kaelbling's interval estimation. We find that Nancy and DDNancy generally outperforms previous methods, particularly on more difficult problems. This work illustrates how metareasoning about uncertainty during planning can be beneficial even in a deterministic setting.

This work appeared in the proceedings of AAAI-20 (Fickert, Gu, Staut, Ruml, Hoffmann, & Petrik, 2020c) and PRL-21 (Fickert, Gu, Staut, Lekyang, Ruml, Hoffmann, & Petrik, 2020a) (All authors did joint work on algorithm design and theoretical analysis. Tianyi Gu implemented the algorithm in the domain-specific framework and conducted experiments in traditional search domains. Sai Lekyang implemented the THTS-WA* algorithm. Maximilian Fickert and Leonhard Staut implemented the algorithm in the domain-independent framework and conduct experiments in

classical planning domains. Maximilian Fickert proved the completeness of Nancy).

2.1 Introduction

Some applications of AI are subject to real-time constraints, where the agent must select its next action within a fixed time bound. Typical examples include user interfaces or the control of cyber-physical systems, where unbounded pauses between system actions are undesirable or potentially dangerous. Real-time planning methods tackle this problem setting. The planning agent in this setting incrementally plans toward a goal, trying to minimize the total cost of the resulting trajectory. Many real-time heuristic search methods follow the basic three-phase paradigm set down in the seminal work of Korf (1990):

1) starting at the agent's current state, expand a fixed number of nodes according to the given time bound to form a lookahead search space (LSS);

2) use the heuristic values of the frontier nodes in combination with the path costs incurred to reach them to estimate the cost-to-goal for each currently-applicable action, and commit to the action with the lowest estimate;

3) to prevent the agent from cycling if it returns to the same state in the future, update the heuristic values of one or more states in the LSS.

For example, in the popular and typical algorithm LSS-LRTA* (Koenig & Sun, 2008), the lookahead in step 1) is performed using A* (Hart et al., 1968), the value estimates in step 2) are implicitly calculated for each node as the minimum f value among its successors (the 'minimin' backup), and the learning in step 3) is performed by updating h values for all nodes in the LSS using a variant of Dijkstra's algorithm.

While elegant and often successful, this paradigm does not explicitly address the uncertainty inherent in real-time planning. As the search computes only a miniscule fraction of the state space (up to the LSS frontier), it must commit to action decisions subject to uncertainty about the part of the state space beyond that frontier. In this chapter, we pursue a view of real-time planning as a form of decision-making under uncertainty.

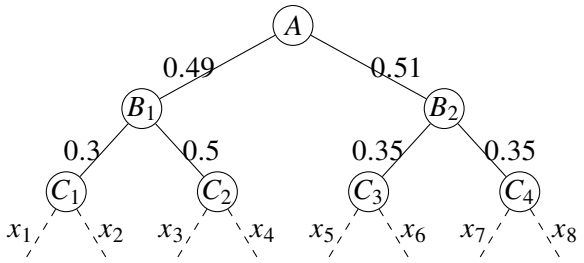


Figure 2-1: Should an agent at A move to B_1 or B_2? (Pemberton & Korf, 1994)

Pemberton and Korf (1994) observed that, given an LSS, it is not in general the best possible decision to head toward the frontier node with the lowest f value. Figure 2-1 shows their example LSS. The agent here is located at node A, and must decide whether to commit to the applicable action transitioning to node B_1, or whether to commit to the other applicable action transitioning to node B_2. The agent knows that all nodes at depth three, i.e. the children of the nodes C_i, are goal nodes. While the true costs x_i of the edges reaching the goal nodes are beyond the agent's lookahead depth and hence unknown, the agent knows that they are uniformly distributed between 0 and 1. The only admissible heuristic cost-to-go estimate for the nodes C_i is $h = 0$.

By the principle of rationality, the agent should take an action that minimizes expected cost. But this is not done in typical real-time search algorithms. The node with the lowest f-value is C_1 (whose value is 0.79), so the typical algorithm would move to B_1. Yet moving to B_2 yields the better expected cost. Note that, at next step, lookahead will reveal the x_i values below B_1 or B_2. As detailed by Pemberton (1995) (in their eq. 1) the expected minimum of $x_5, ..., x_8$ is 0.2, yielding a total expected cost of 1.06 for B_2, while the total expected cost of B_1 is 1.066. In other words, having many good-looking options can be statistically better than a single great-looking option. This problem does not arise in the off-line planning setting, as there the agent can discover all relevant edge costs before committing to an action.

Furthermore, as pointed out by Mutchler (1986), A*'s policy of expanding the frontier node with the lowest f value is not, in general, the optimal way to make use of a limited number of node expansions. If we view the agent as facing a decision under uncertainty, it is sometimes beneficial to gain knowledge about inferior-looking options.

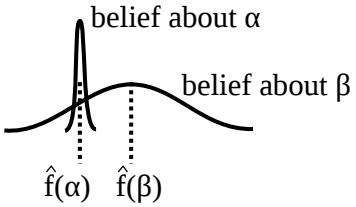


Figure 2-2: Should an agent expand nodes under α or β?

Consider the situation depicted in Figure 2-2 for illustration. The figure shows the agent's current beliefs about the expected total plan cost that would be incurred by committing to the applicable actions α and β respectively. Each such belief is a probability distribution over possible costs, with the expected value denoted by $\hat{f}(\cdot)$. In the displayed situation, the agent is quite certain about the value of α but quite uncertain about the value of β. It is likely that α is better, but there is a significant possibility that β may be better instead. Given this, expanding frontier nodes under α can be less useful than expanding under β, even though β is believed to have a higher expected cost. It can be more important to explore the possibility that a poorly-understood option might in fact be great than to nail down the exact value of a good-looking option that is already well understood. Note that the problem faced by real-time search is how best to spend a given number of expansions.

This is distinct from the well-known exploration-versus-exploration dilemma, as we are given the exact amount of time that we should spend exploring. We also note that this problem does not arise in off-line optimal search, where every node whose f value is less than the optimal solution cost must be expanded.

Although these insights derived from a reasoning under uncertainty perspective seem powerful, they have not yet resulted in generally-applicable real-time heuristic search algorithms. In this chapter, we investigate a recently proposed real-time heuristic search framework , called Nancy, that uses explicit reasoning about the uncertainty stemming from the unknown parts of a deterministic state space. Nancy treat the traditional heuristic estimate of cost-to-go from a given state as uncertain evidence, inducing beliefs – probability distributions – over the actual remaining cost. We will focus on how to represent such beliefs.

In the original Nancy, the beliefs are assumption-based, assuming Gaussian distributions as in

prior work (O'Ceallaigh & Ruml, 2015). As these assumptions are not necessarily justified, in this work, we introduce an alternative data-based approach, approximating the beliefs from data previously gathered in the same problem family.

We evaluate our methods on a collection of search benchmark domains. We explore the effect of lookahead size and of assumption-based vs. data-based beliefs, and we compare to the state of the art in real-time planning. We also compare to previous work on exploiting uncertainty in search. Monte-Carlo tree search (MCTS) methods (Kocsis & Szepesvári, 2006; Keller & Helmert, 2013; Feldman & Domshlak, 2014; Silver, Hubert, Schrittwieser, Antonoglou, Lai, Guez, Lanctot, Sifre, Kumaran, Graepel, Lillicrap, Simonyan, & Hassabis, 2018), which originate in probabilistic planning, maintain node-value averages along with node-visited counts and use these to give a boost to actions with uncertain values. The previous work perhaps closest to ours, Interval Estimation (Kaelbling, 1993), explicitly represents uncertainty and uses it to guide search effort. We adapt these methods to our setting. Overall, we find that DDNancy and Nancy typically outperform previous methods despite their metareasoning overhead, suggesting that they makes better use of limited node expansions. The success of Nancy and DDNancy illustrates the strength of adopting a reasoning under uncertainty perspective for resource-bounded decision-making, even in completely deterministic problem domains.

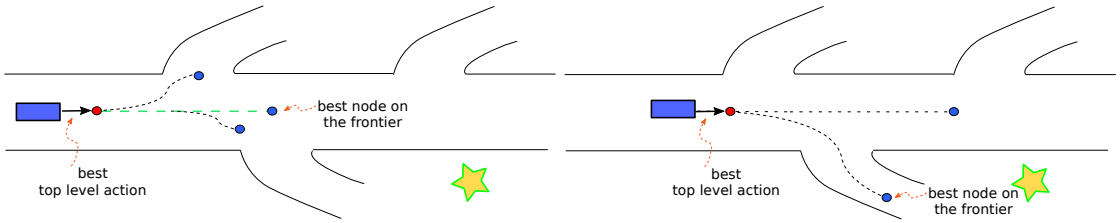
2.2 Background

In this section, after describing the problem setting of real-time heuristic search, we touch on the main areas of related work.

2.2.1 Problem Definition

Real-time Search

Systems that interact with the external physical world often must be controlled in real time. Examples include systems that interact with humans and robotic systems, such as autonomous vehicles. In this chapter, we address real-time planning, where the planner must return the next action for the



(a) (b)

Figure 2-3: Iterations of LSS-LRTA*.

system to take within a specified fixed time bound. We assume the system has fully observable state, discrete time, and discrete deterministic actions. We adopt a heuristic state-space search approach: while the system transitions from state s_1 to s_2, the planner exploits a heuristic cost-to-go function to explore promising parts of the state-space graph starting from s_2 to find an appropriate action to execute once the transition completes. The duration of the transition gives rise to the real-time bound for the planner. The problem domain is represented by an initial starting state, a successor state generator, and a goal predicate on states.

Real-time search adheres to a given hard real-time constraint by performing a bounded lookahead. The agent expands the search space until a time bound is reached, at which point the agent will then take either a single step or a sequence of steps towards the edge of the local search space that was expanded. While this partial plan is being executed, the search computes the next actions to take. In this way, the agent incrementally constructs a solution path.

2.2.2 Related Work

There have been many proposals for real-time heuristic search methods. Some, like LSS-LRTA*, are very general and apply to any state space search problem. Others assume undirected state spaces in which it is always possible to immediately return to a node's parent state. Several are specialized to grid-based pathfinding. In this paper, we choose LSS-LRTA* as our point of comparison due to its simplicity and generality.

LSS-LRTA*

Recall from Section 2.1 that LSS-LRTA* performs an A* lookahead, limited by a fixed number of nodes to expand (Koenig & Sun, 2008). The progress of LSS-LRTA* shown on Figure 2-3. Intuitively, this is superior to a simple breadth first search towards a fixed depth because it takes the heuristic into account. LSS-LRTA* also performs Dijkstra's algorithm to update the h-values of all nodes in the local search space. Dijkstra's algorithm is started from the open list of the previous lookahead search. The algorithm terminates when the goal is expanded during the A* expansion stage.

Single Step LSS-LRTA* (LSS-LRTA* SS) is a variation of LSS-LRTA* in which the agent takes only a single step on the path that A* generated, rather than committing to an entire path (Burns, Kiesel, & Ruml, 2013). Only taking a single step allows LSS-LRTA* to update the heuristic values of nearby nodes more often. This creates a tradeoff between execution time and action confidence.

Mutchler (1986) raises the question of how best to allocate a limited number of expansions. His analysis considers complete binary trees of uniform depth where each edge is randomly assigned cost 1 with probability p and cost 0 otherwise. He proves that a minimum f expansion policy is not optimal for such trees in general, but that it is optimal for certain values of p and certain numbers of expansions. It is not clear how to apply these results to more realistic state spaces.

Pemberton and Korf (1994) point out that it can be useful to use different criteria for action selection versus node expansion. They use binary trees with random edge costs uniformly distributed between 0 and 1 and use a computer algebra package to generate code to compute exact $\hat{f}$ values under the assumption that only one or two tree levels remain until a goal (the 'last incremental decision problem'). As we will discuss in more detail below, this requires representing and reasoning about the distribution of possible values under child nodes in order to compute the distribution at each parent node. They conclude that a strategy based on expected values is barely better than the classic minimin strategy and impractical to compute for state spaces beyond tiny trees. They also investigate a method in which the nodes with minimum f are expanded and the action with minimum $\hat{f}$ is selected and find that it performs better than using f for both.

Given the pessimism surrounding exact estimates, Pemberton (1995) proposes an approximate

method called k-best. Only the k best frontier nodes below a top-level action are used to compute its value, allowing a fixed inventory of equations derived in advance to be used to compute expected values during search. Although this approach did surpass minimin in experiments on random binary trees, Pemberton concludes that its complexity makes it impractical. It is also not clear how to apply these results beyond random binary trees.

Mo'RTS

Our problem setting bears a superficial similarity to the exploration/exploitation trade-off examined in reinforcement learning. However, note that our central challenge is how to make use of a given number of expansions — we do not have to decide between exploring for more information (by expanding additional nodes) or exploiting our current estimates (by committing to the currently-best-looking action). DTA* (Russell & Wefald, 1991) and Mo'RTS (O'Ceallaigh & Ruml, 2015) are examples of real-time search algorithms that directly address that trade-off. Both are based on estimating the value of the information potentially gained by additional lookahead search and comparing this to a time penalty for the delay incurred. DTA* expands the frontier node with minimum f and Mo'RTS expands the frontier node with minimum $\hat{f}$. However, neither DTA* nor Mo'RTS take into account the uncertainty of their estimates during the expansion phase.

MCTS

MCTS algorithms such as UCT (Kocsis & Szepesvári, 2006) share our motivation of recognizing the uncertainty in the agent's beliefs and trying to generate relevant parts of the state space. Tolpin and Shimony (2012) emphasize the purpose of lookahead as aiding in the choice of the agent's next action and, as we will show below, they take an approach motivated by the value of information. Lieck and Toussaint (2017) and Shperberg, Shimony, and Felner (2017) investigate selective sampling for MCTS. However, unlike most work in MCTS, we focus on deterministic problems and we have no need to sample action transitions or perform roll-outs. Furthermore, real-time planning can arise in applications where perhaps only a dozen nodes can be generated per decision, a regime where MCTS algorithms can perform poorly, as a single roll-out may generate hundreds of nodes.

Bayesian reinforcement learning

Work on active learning also emphasizes careful selection of computations to refine beliefs. For example, Frazier, Powell, and Dayanik (2008) present an approach they term 'the knowledge gradient' for allocating measurements subject to noise in order to maximize decision quality. More broadly, the notion of representing beliefs over values during learning and decision-making has been pursued in Bayesian reinforcement learning (Bellemare et al., 2017).

2.2.3 General Nancy : A New Real-time Search Framework

In this subsection we give a complete description of Nancy, a recently proposed real-time search algorithm which can be thought of as an instantiation of LSS-LRTA* using different real-time search components. Nancy emerges as a combination of a risk-based lookahead search which takes into account the uncertainty of its estimates, together with the eponymous Nancy backups.

Assembling Nancy

Algorithm 1 shows the pseudo-code for Nancy. The lookahead uses risk to guide the direction of the lookahead, building up the local search space (LSS) and returning the overall best frontier node according to $\hat{f}$ (line 4), breaking ties by $\hat{h}$. The backup function is used to update the beliefs by backing up the beliefs from the frontier of the local search space towards the root.

Risk-based Lookahead

We now explain Nancy's lookahead strategy in more detail.

Lookahead is performed to minimize risk. The idea is to find the optimal action to execute next while also becoming more certain about possible alternative actions. Algorithm 2 shows the pseudo-code for the risk-based lookahead. In the lookahead phase, Nancy uses her first expansion to generate the top level actions (line 1). From that point forward, Nancy expands nodes such that an approximation of risk is minimized, until the expansion or time limit of the lookahead runs out. Each top-level action has an associated open list (denoted by choosen TLA.open in Algorithm 2, line 8) that is ordered by $\hat{f}$. Before each expansion, Nancy has to pick the open list where the expansion

Algorithm 1: Nancy

$s \coloneqq s_{start}$

$\pi_{curr} \coloneqq \langle\rangle$

while $s[\![\pi_{curr}]\!]$ *is not a goal state* **do**

    $t \coloneqq$ `risk_lookahead`(s)

    $\pi_{curr} \coloneqq$ `update_path`(s, t, π_{curr}, π)

    `apply_next`(s, π_{curr})

    `backup`(lss)

while *s is not a goal state* **do**

    `apply_next`(s, π_{curr})

fn `apply_next`(s, π_{curr})

    let $a_0, \ldots, a_n$ be the action sequence of π_{curr}

    $s \coloneqq s[\![a_0]\!]$

    $\pi_{curr} \coloneqq \langle a_1, \ldots, a_n \rangle$

fn `update_path`(s, t, π_{curr}, π)

    if $(s[\![\pi_{curr}]\!]$ *was expanded in the lookahead* or

        t *is a goal state* or

        $\hat{f}(t) < \hat{f}(s[\![\pi_{curr}]\!])$ or

        $\hat{f}(t) = \hat{f}(s[\![\pi_{curr}]\!])$ and $\hat{h}(t) < \hat{h}(s[\![\pi_{curr}]\!]))$ **then**

        return π

    return π_{curr}

Algorithm 2: Risk-Based Lookahead

Input: s : state

Output: t : target state with minimal $\hat{f}$

Generate *TLAs*

while *lookahead limit is not reached* **do**

  for *tla in TLAs* **do**

    Swap in $\mathcal{B}_{post}(s[\![tla]\!])$ for $\mathcal{B}(s[\![tla]\!])$

    $risk[s[\![tla]\!]] \coloneqq \texttt{risk}(TLAs)$

    Restore original $\mathcal{B}(s[\![tla]\!])$

  $chosen \coloneqq \underset{tla \in TLAs}{\arg\min}\ (risk[tla])$

  $t \coloneqq chosen.open.pop\_min()$

  if *t is goal* **then**

    return t

  for $a \in A(t)$ **do**

    Estimate and cache $\mathcal{B}(t[\![a]\!])$ and $\mathcal{B}_{post}(t[\![a]\!])$

    $\texttt{push}(chosen.open, t[\![a]\!], \hat{f}(t[\![a]\!]))$

  $u \coloneqq chosen.open.min()$

  $\mathcal{B}(s[\![chosen]\!]) \coloneqq \mathcal{B}(u) + g(u)$

  $\mathcal{B}_{post}(s[\![chosen]\!]) \coloneqq \mathcal{B}_{post}(u) + g(u)$

$best \coloneqq \underset{tla \in TLAs}{\arg\min}\ (\hat{f}(s[\![tla]\!]))$

return $best.open.min()$

will take place. For this purpose, Nancy estimates $\mathcal{B}_{post}$ which denotes the updated belief Nancy expects after performing one expansion. The open list where Nancy expects to arrive at a belief with minimal risk is then selected and the actual expansion follows, (Algorithm 2, line 11). After each expansion, new information is obtained about the frontier of the local search space. To make use of this information, the belief at the top level needs to be updated, such that it agrees with the new best frontier node(Algorithm 2, line 15 and 16).

This lookahead process is repeated until the expansion or time limit is reached, or a goal state is selected for expansion. Once the lookahead phase ends, the search performs Nancy backups and executes the TLA with the lowest expected cost (Algorithm 1, line 12). In the learning phase, the beliefs $\mathcal{B}$ and post-expansion beliefs $\mathcal{B}_{post}$ of all nodes within the local search space are updated (Algorithm 1, line 7). This learning process performs a dynamic programming-like learning step to update the $\hat{h}$-values of the expanded states (like LSS-LRTA*).

Nancy Assumptions

As presented by Mitchell et al. (2019), Nancy makes use of at least four assumptions. First, she assumes beliefs about the true h^* values of frontier nodes that are Gaussian, centered around an $\hat{h}$ estimate derived from h and an error model estimated on-line during search. The error model assumes a constant underestimate by h for every edge in the state space graph and requires an estimate of the number of edges-to-go. The edges-to-go are estimated by h in unit-cost domains or a similar function d in domains with non-uniform costs. The variance was assumed to be proportional to the edges-to-go as well.

Second, Nancy back-propagates only the belief of the successor with the lowest expected value. As Mitchell et al. (2019) explain, this assumes that no more information will become available to distinguish between sibling nodes. And if multiple successors look promising, they should each be allowed to influence our belief about their parent.

Third, the regret calculation requires estimating the change in belief that would be caused by expanding nodes under a top-level action. While this is likely complex, Nancy assumes it to be a simple reduction in variance of the current belief, proportional to the estimated reduction in the

number of edges-to-go. In fact, done properly, the post-expansion belief should be a distribution over possible beliefs, and just as with the backups, multiple successors should be allowed to influence the estimate.

And fourth, in her regret calculation, Nancy does not consider the combinatorial space of outcomes, but just considers two top-level actions at a time and assumes independence of subtrees.

While some of these assumptions may be necessary for computational efficiency and model simplicity, in the next session of this dissertation we begin the process of understanding how to weaken them by showing how the first assumption, regarding the leaf belief distributions, can be replaced by data.

2.3 Data-Driven Nancy

How to obtain beliefs about the distance to the goal is a principal concern of Nancy. For this purpose, Nancy makes use of several assumptions about heuristic behavior. In this chapter we describe an alternative approach to obtain these beliefs, which is based on data. The idea of this method is to use statistics about heuristic behavior gathered in an offline phase prior to the search to construct the beliefs. Hence, some or all of the assumptions to construct beliefs at runtime are replaced with data. Here we cover the details of the data-driven variant of Nancy, which we call DDNancy.

2.3.1 Data Generation

The purpose of Nancy's assumptions is to obtain a better estimate of the possible true cost-to-go $h^*(s)$ when only $h(s)$ is available. The approach to replace these assumptions is to run an offline training phase that learns the distribution of h^*-values.

Algorithm 3 shows a high-level overview of the data collection process. The h^* distributions are generated offline by collecting (h, h^*) pairs from a number of training instances. Each training instance is first solved by an initial search. Each state that was expanded by that search is then solved optimally, and its h and h^* values are stored. By collecting all these pairs, we obtain a set of h^* values for each h value, making up the distribution.

In its search, DDNancy uses the heuristic values to look up the corresponding distribution of

Algorithm 3: Data Collection

```
Pick a set of training problem instances 𝒯
Pick a search algorithm 𝒮
for t in 𝒯 do
    Solve t with 𝒮, while recording all expanded states
    for state s expanded by 𝒮 do
        Solve s optimally
        Store pair (h(s), h*(s))
```

h^* values. It may happen that DDNancy encounters a state with a heuristic value h that was not observed in the data gathering process. In that case, we perform an online extrapolation step on the data. We pick the largest $h' \leq h$ for which we have a distribution in the data set. The distribution for h is extrapolated by shifting the distribution of h' by the difference between h and h', i.e., adding $(h - h')$ to each data point in the distribution of h'. The newly created distribution is cached to have it available for the remaining search.

In the learning step, the data-driven instantiation of Nancy does not change the heuristic values. Instead, the change in heuristic value is transferred to the distributions, and the data points are shifted accordingly (similar to the extrapolation procedure). In the implementation, we simply store the current shift value and a pointer to the corresponding distribution for each expanded state.

In the training process, the initial search is used to sample the states for the data collection (as the expanded states are then solved optimally). There are two key motivations to use this strategy instead of training on the entire state space. The first reason is practical feasibility: For instances above a certain size , solving the entire state space would require an unreasonable amount of time and memory. The second reason is that considering the entire state space may not make the data more accurate, as most of these states are not seen in the actual search. Instead, we want the process to focus on states that are representative for states encountered by Nancy, and inform the algorithm how the heuristic typically behaves on such a state. Since DDNancy is a suboptimal search algorithm,

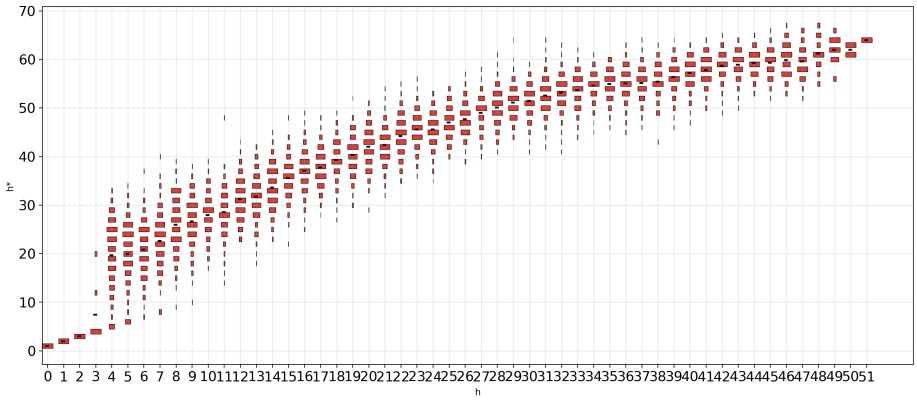


Figure 2-4: Beliefs generated for uniform-cost 15-puzzle.

we also use a suboptimal search algorithm to sample the states for the data generation. The initial search algorithm should therefore have similar behavior to Nancy to generate a good set of sample states, and improve the accuracy of the resulting distributions. We generate the data for each domain separately. While this requires additional work for each new domain DDNancy is intended to run on, as we will see, the heuristic behavior can vary a lot between different domains, and domain-specific data can capture the behavior more accurately.

2.3.2 Generated Data

In general, the generated beliefs with weighted A* and LSS-LRTA* are very similar, with only minor differences for large heuristic values where fewer samples have been observed. This is somewhat expected; while the two algorithm expand different sets of states due to their different expansion strategies, the underlying instances are the same, and the algorithms will find similar solutions. We conjecture that it is unlikely for them to observe a very different heuristic behavior over their respective sets of states. For the remaining experiments (and the DDNancy results shown in the previous section) we use weighted A* with a weight of 2 to sample the training states.

We show the belief distributions that result from using weighted A* with a weight of 2. The x-axis shows the h-value and the y-axis shows the h* distribution. In Figure 2-4, the expected value of the distributions on the 15-puzzle (detail introduced in the next subsection) makes a large jump at $h = 4$. This shows the potential inaccuracy of the Manhattan distance heuristic in the 15-puzzle:

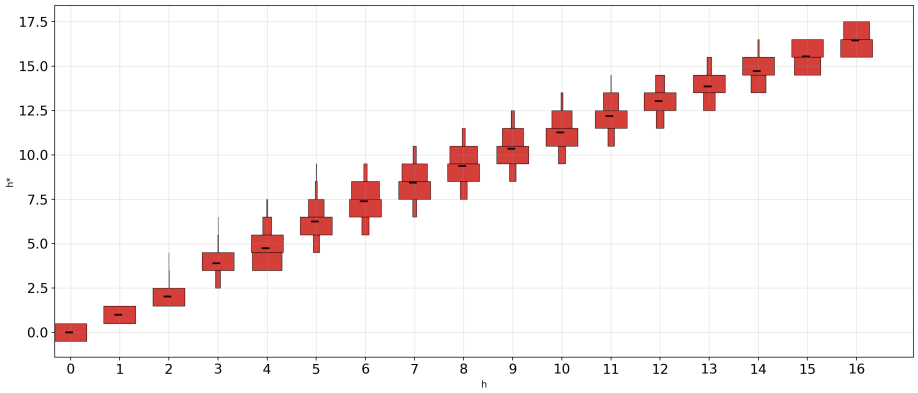


Figure 2-5: Beliefs generated for 16-pancake.

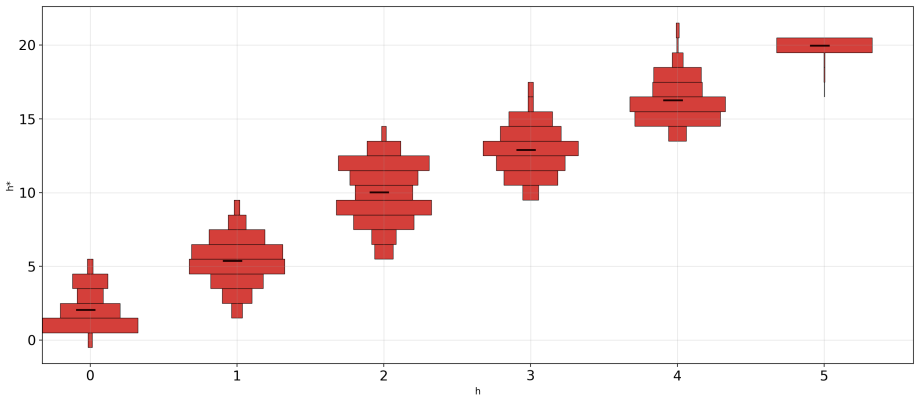


Figure 2-6: Beliefs generated for Barto Racetrack.

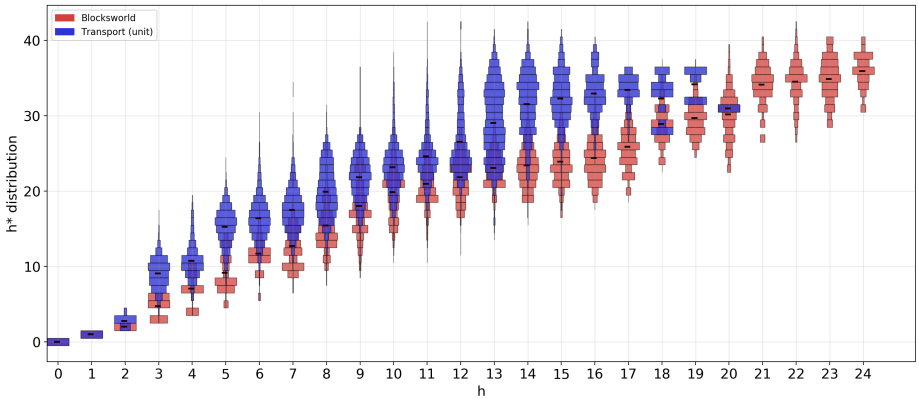


Figure 2-7: Beliefs gathered for the Blocksworld and Transport (unit-cost).

there are states where the heuristic value is small, but an optimal solution still requires a significant number of moves.

Figure 2-5 and Figure 2-6 show examples of the generated data for pancake (detail introduced in the next subsection) with gap heuristic and racetrack with Dijkstra heuristic to demonstrate the differences in heuristic behavior on these domains. The distributions are very smooth for the pancake puzzle and the racetrack domain, however, the shapes are different. For pancake domain, we have small variance distribution confers with the well known knowledge that gap heuristic is a very accurate heuristic for pancake problem, while in racetrack, we get higher variance beliefs with multi-model shapes indicate the Dijkstra heuristic is not as accurate.

A similar comparison of the generated data for the classic planning domains is shown in Figure 2-7, generated by Leonhard Staut. The expected value increases roughly monotonically in both domains, but slightly faster in Transport, where the expected value makes a jump when going from $h = 2$ to $h = 3$. Furthermore, the variance of observed h^* values is greater in Transport. In Blocksworld, the training process encountered states with slightly larger heuristic values than those in Transport.

Completeness

In this section, we show that Nancy is complete. We give a completeness proof for a more general class of real-time search algorithms, and then show that Nancy is a member of this class. Our proofs

use $\hat{h}$ to denote the updated heuristic, which, for real-time algorithms based on belief distributions such as Nancy, is the expected value of the (updated) belief distributions.

We assume that: (A1) action costs are strictly positive, (A2) for every state, there is a goal reachable from it, (A3) all initial beliefs have finite expected value, (A4) the state space is finite. Our proof follows the style of Korf's (1990) proof for RTA* and Bulitko and Sampley's (2016) proof for Weighted Lateral LRTA* (wbLRTA*): we first prove that incompleteness implies that there must exist a subset of states within which the agent circulates forever. Then we prove that there cannot exist such a set due to the updates made by the real-time search algorithm's learning rule.

Definition 8 *In a real-time search algorithm, a subset of states $S_\circ$ is called a* circulating set *if there exists a time $t_\circ$ after which the agent will visit only states $s \in S_\circ$ and visit each one an infinite number of times.*

Lemma 1 *Under assumptions (A2) and (A4), if a real-time search algorithm is incomplete, it must have a circulating set $S_\circ$.*

Proof: Since a goal is reachable from all states (A2)[1], a real-time search only terminates when it reaches a goal state, so incompleteness means that the search never terminates. Because the state space is finite (A4), there must exist a subset of non-goal states $S_\circ$ such that the agent will re-visit each of the states in $S_\circ$ an infinite number of times after some initial time t. Let S be the set of non-goal states. If there exist states $s \in S$ that are not visited an infinite number of times, let the last time such a state s is visited be t_s. Then $t_\circ := \max(t, t_s)$ satisfies the claim. □

Definition 9 *A real-time search is called* goal aware *if, upon generating a goal state in its lookahead, it commits to the path towards it.*

Nancy is goal aware (see Section 2.2) and so is LSS-LRTA* (Koenig & Sun, 2008, Figure 5, line 32).

Lemma 2 *Under assumptions (A2) and (A4), if a goal-aware real-time search algorithm has a circulating set $S_\circ$, then a) there exists a finite set of non-goal states $S_\infty \supseteq S_\circ$ that are expanded*

[1]Each non-goal state having at least one successor would be sufficient for this argument. This is implied by assumption (A2), which we use instead for simplicity.

infinitely often, b) every successor state $s' \in S_F := \{s[\![a]\!] \mid s \in S_\infty, a \in A(s)\} \setminus S_\infty$ *appears infinitely often in the frontier of lookaheads from states in* $S_\circ$*, c) there is a time* t_1 *after which no* $s' \in S_F$ *is expanded, and d)* S_F *is non-empty.*

Proof: The lookaheads from all states $s \in S_\circ$ are performed infinitely often with a fixed number of expansions, so for each $s \in S_\circ$ there must be a set of states $S^s_\infty \supseteq \{s\}$ that are expanded infinitely often, and so is their union S_∞. S_∞ cannot contain a goal, as a goal-aware search would head towards it, breaking the circulation. The neighbor states S_F are obviously generated infinitely often from S_∞; as they are not in S_∞, they are expanded only a finite number of times so the claimed time t_1 exists. S_F is non-empty because S_∞ does not contain a goal state, but the goal is reachable from all states (A2), so S_F must contain at least one state on a path to the goal. All of these sets must be finite since the state space is finite (A4). □

Definition 10 *A learning algorithm is called* dynamic programming-like *if it updates the heuristic values of the states S expanded in the local search space (and no others) such that afterwards the heuristic values of all* $s \in S$ *are* locally consistent*, that is, satisfy*

$$\hat{h}(s) = \min_{a \in A(s)} (c(a) + \hat{h}(s[\![a]\!])).$$

A standard result is that, if updates of the form $\hat{h}(s) := \min_{a \in A(s)}(c(a) + \hat{h}(s[\![a]\!]))$ are performed infinitely often on a finite state-space graph with positive action costs and reachable goals, starting from arbitrary finite initial values the state values will eventually converge to locally consistent values (Bertsekas & Tsitsiklis, 1996, Proposition 2.3).

Lemma 3 *Under assumptions (A1)-(A4) with* S_∞ *as in Lemma 2, if a goal-aware real-time search algorithm that performs dynamic programming-like learning has a circulating set* $S_\circ$*, then there exists a time* t_2 *after which, for every* $s \in S_\infty$*,* $\hat{h}(s)$ *will be locally consistent.*

Proof: Consider the state space sub-graph S' induced by $S_\infty \cup S_F$ (Lemma 2). After time t_1 as per Lemma 2, the update operation is performed infinitely often on all states $s \in S_\infty$, and only on those states. All update operations are well defined, i.e., consider successors contained in S'. Due to the convergence of the dynamic programming-like learning procedure with positive action costs (A1)

and bounded initial $\hat{h}$ values (A3), the $\hat{h}$ values of all states $s \in S_\infty$ will eventually converge to a solution of the state-update equation as claimed. □ In the following, we will use f_s and g_s to denote the f value and g value, respectively, of a state in a lookahead search space with respect to the current root state s of the lookahead.

Lemma 4 *Under assumptions (A1), (A2), and (A4), if $\hat{h}$ is locally consistent on all states S_E^s expanded in a lookahead search space rooted at a state s that does not contain a goal, then all states s' on a cheapest path within the lookahead from s to a frontier node with minimal $\hat{f}_s$ have $\hat{f}_s(s') = \hat{f}_s(s)$.*

Proof: Let S_F^s be the frontier nodes of the lookahead from s. Due to assumption (A2), S_F^s must be non-empty since S_E^s does not contain a goal state. Let π be a path from s to the frontier such that each step satisfies the state-update equation (i.e., taking the arg min according to Definition 10). Such a path must exist because the lookahead search space is finite due to assumption (A4) and the heuristic is locally consistent on each state within the lookahead. Furthermore, due to positive action costs (A1), the path can not contain cycles and hence must eventually reach the frontier. Each state s' on π has $\hat{f}_s(s') = \hat{f}_s(s)$ by construction. This includes the state $s[\![\pi]\!] \in S_F^s$, thus $\hat{f}_s(s)$ is an upper bound on the minimal $\hat{f}_s$ value among the frontier nodes. Observe that, due to local consistency, $\hat{f}_s$ can only increase on any path from s to the frontier. Therefore, there can not be a state $s_f \in S_F^s$ with $\hat{f}_s(s_f) < \hat{f}_s(s)$. Furthermore, for all states $s_f \in S_F^s$ with $\hat{f}_s(s_f) = \hat{f}_s(s)$, there must be a path from s to s_f such that all states s' on that path have $\hat{f}_s(s') = \hat{f}_s(s)$ as claimed. □ In particular, Lemma 4 holds in each lookahead search space that does not contain a goal after the corresponding learning phase of a real-time search algorithm that performs dynamic-programming-like learning. Furthermore, it holds for every lookahead search space that only expands states $s \in S_\infty$ after time t_2 as per Lemma 3.

Definition 11 *A real-time search algorithm is called* reasonable*, if it (1) is goal aware, (2) uses dynamic-programming-like learning, and (3) performs action selection by $\hat{f}$, i.e., unless the search already discovered a goal which it is moving towards, after the lookahead it applies the first action of the cheapest path (within the lookahead) towards a frontier node with minimal $\hat{f}$ value.*

Lemma 5 *Under assumptions (A1)-(A4), a reasonable real-time search algorithm cannot have a circulating set.*

Proof: We proceed by contradiction. Assume that the search algorithm does have a circulating set $S_\circ$. Then there must be sets S_∞ and S_F as per Lemma 2. After some point in time t_2 as per Lemma 3, $\hat{h}$ is converged to locally consistent values on all states $s \in S_\infty$. Let $s = \arg\min_{s \in S_\circ} \hat{h}(s)$ be a state from the circulating set with minimal $\hat{h}$ value after convergence. Since s is visited infinitely often, there must be a time $t_3 > t_2$ when a lookahead originates from s.

Let $s_f = \arg\min_{s_f \in S_F^s} \hat{f}_s(s_f)$ be a frontier node with minimal $\hat{f}_s$, and let $s, s_1, \ldots, s_f$ be the states along the cheapest path in the lookahead to s_f. According to Lemma 4, the $\hat{f}_s$ values on this path must all be equal, i.e., $\hat{f}_s(s) = \hat{f}_s(s_1) = \cdots = \hat{f}_s(s_f)$. Due to the assumption of positive action costs (A1), this implies $\hat{h}(s) > \hat{h}(s_1) > \cdots > \hat{h}(s_f)$. Following action selection by $\hat{f}$, the agent will make a step towards s_f by moving to s_1 after the lookahead, implying that $s_1 \in S_\circ$. Since s was selected to have minimal $\hat{h}$ value among all states in $S_\circ$, we have a contradiction with $\hat{h}(s_1) < \hat{h}(s)$. □

Theorem 1 *Under assumptions (A1)-(A4), a reasonable real-time search algorithm will eventually reach a goal.*

Proof: If the search never reaches a goal, it has a circulating set by Lemma 1, which by Lemma 5 is not possible. □

We next show that Nancy is a reasonable search algorithm according to Definition 11, which allows us to prove its completeness. However, in order to prove that Nancy does action selection by $\hat{f}$ even when using persistence, we first need the following lemma.

Lemma 6 *Under assumptions (A1), (A2), and (A4), if Nancy does not switch to a new path after the lookahead from a state s (i.e., does not go into the* **then** *branch in the* `update_path()` *function, see Algorithm 1, line 14), then a) π_{curr} must lead through the frontier, b) the state of π_{curr} that lies on the frontier has minimal $\hat{f}_s$ value among all frontier nodes, c) $\hat{f}_s(s') = \hat{f}_s(s)$ for all states s' on π_{curr}, and d) the $\hat{h}$ values of these states remain unchanged in the learning phase.*

Proof: Let s_l be the root state of the lookahead when π_{curr} was set. In that lookahead, $s_l[\![\pi_{curr}]\!]$ was the frontier node with minimal $\hat{f}_{s_l}$ value, and, after the learning phase, all states s' on π_{curr} have equal $\hat{f}_{s_l}$ according to Lemma 4. When Nancy moves along π_{curr} in the action selection phase, the $\hat{f}$ values relative to the current root state are equally reduced by the cost of the applied action for all remaining states along π_{curr}. Thus, only the heuristic update in the learning phase could invalidate the claim that all states along the path have equal $\hat{f}_s$ value.

If Nancy does not switch to a new path after a lookahead rooted at s, then we know that (1) $s[\![\pi_{curr}]\!]$ was not expanded, so π_{curr} must lead through the frontier, and (2) there is no state s_f on the frontier of the lookahead with $\hat{f}_s(s_f) < \hat{f}_s(s[\![\pi_{curr}]\!])$. Before the learning phase, all states s' on π_{curr} have $\hat{f}_s(s') = \hat{f}_s(s)$. Since one such state s' is on the frontier (1) and has minimal $\hat{f}_s$ value (2), this still holds after the learning phase according to Lemma 4, and the $\hat{h}$ values can not have changed. □ Now we can prove that Nancy is reasonable as it performs action selection by $\hat{f}$ even when following the path for persistence.

Lemma 7 *Nancy is a reasonable real-time search algorithm.*

Proof: Nancy is goal aware and does dynamic-programming-like learning (Section 2.2). For action selection, Nancy distinguishes two cases: If Nancy does not continue along the cached path, the new path is selected towards a frontier node with minimal $\hat{f}$ value. Otherwise, Nancy follows the path cached for persistence, which, according to Lemma 6, also leads to a frontier node with minimal $\hat{f}$ value. Thus, Nancy does indeed perform action selection by $\hat{f}$ in both cases, and satisfies the definition of a reasonable real-time search algorithm. □

Corollary 1 *Under assumptions (A1)-(A4), Nancy will eventually reach a goal.*

Proof: Nancy is a reasonable real-time search algorithm (Lemma 7), and is thereby complete under assumptions (A1)-(A4) via Theorem 1. □

Theorem 1 also implies that LSS-LRTA* is complete even for heuristics that are not consistent or admissible (a consistent heuristic is the only case proven in the original paper):

Lemma 8 *LSS-LRTA* is a reasonable real-time search algorithm.*

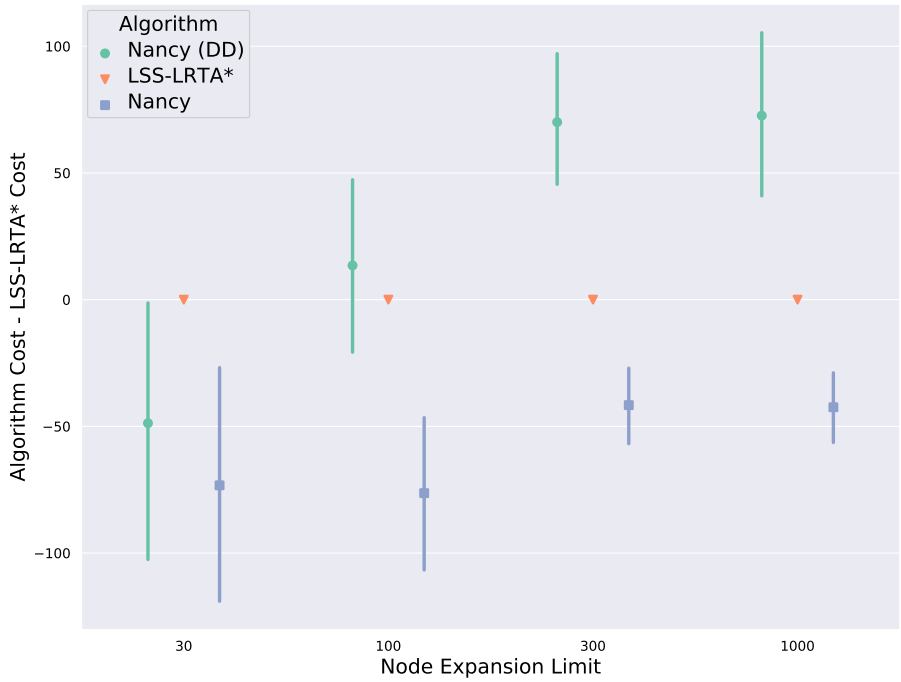


Figure 2-8: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the uniform-cost variants of the 15-puzzle, with error bars indicating 95% confidence intervals.

Proof: LSS-LRTA* is goal aware (Koenig & Sun, 2008, Figure 5, line 32), does dynamic programming-like learning (Koenig & Sun, 2008, Figure 5, line 17-26), and does action selection by $\hat{f}$ (Koenig & Sun, 2008, Figure 5, line 34). □

Corollary 2 *Under assumptions (A1)-(A4), LSS-LRTA* will eventually reach a goal.*

Proof: LSS-LRTA* is a reasonable real-time search algorithm (Lemma 8), and is thereby complete under assumptions (A1)-(A4) via Theorem 1. □

2.3.3 Evaluation

In this section, we compare the Nancy and DDNancy algorithm to LSS-LRTA* on various search domains to show how representing uncertainty and using it to guide real-time search can yield better performance.

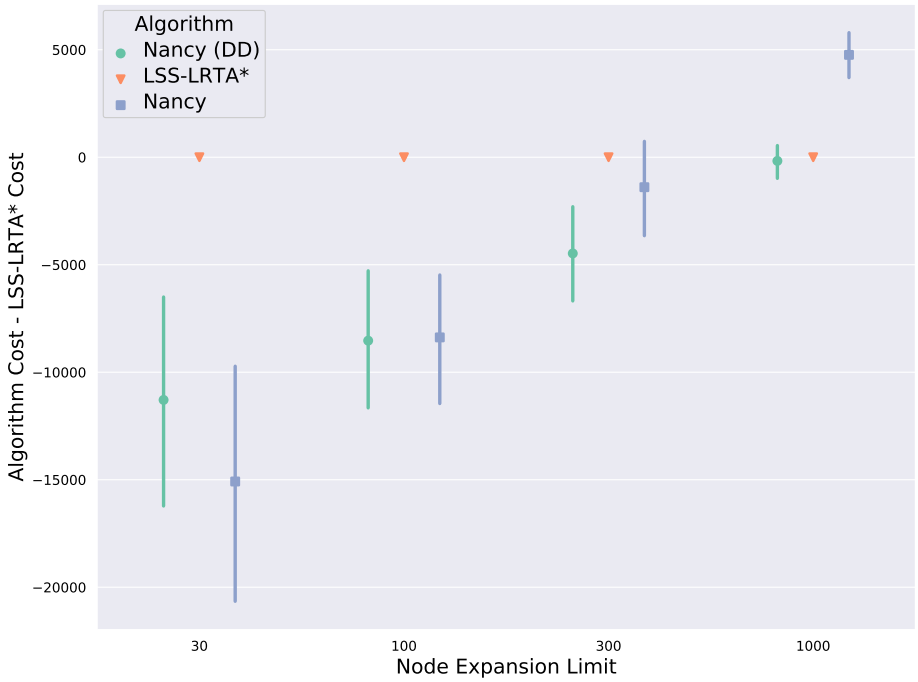


Figure 2-9: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the heavy-cost variants of the 15-puzzle, with error bars indicating 95% confidence intervals.

Sliding-Tile Puzzle

Here we show experiments on the classic 100 15-puzzle instances published by Korf (1985). We test two variants: uniform-cost, in which every action costs one, and heavy, in which the action cost is equal to the label of the moved tile. We use the Manhattan distance heuristic (the sum of all the horizontal and vertical distance from each tile's current location to its goal location) for uniform-cost tile and weighted Manhattan distance for heavy-cost tile.

Figure 2-8 and 2-9 show a comparison of Nancy and DDNancy to LSS-LRTA*. In both 15-puzzle variants, Nancy significantly improves the solution quality compared to LSS-LRTA*. This is particularly the case when the lookahead is limited to only few expansions, and the choice of which nodes should be expanded becomes more important. For larger lookahead bounds the solution costs converge, though LSS-LRTA* performs slightly better on the heavy-cost variant with the largest tested lookahead of 1000 expansions.

DDNancy performs worse than assumption-based Nancy and LSS-LRTA* when using uniform costs, but is the best algorithm in the heavy-tile variant. We conjecture that this is because, due

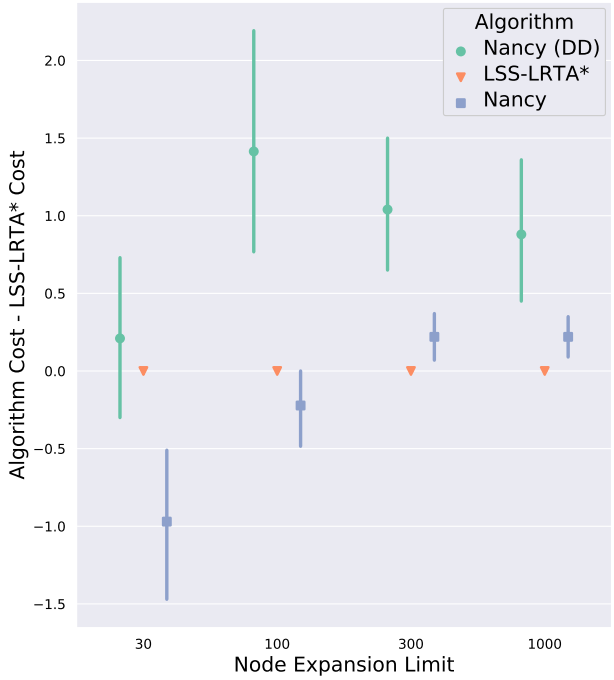


Figure 2-10: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 16 pancakes.

to the popularity of the uniform-cost 15-puzzle as a heuristic search benchmark, the assumptions embodied in Nancy were likely heavily informed by search behavior in this domain, the smoothness of its model may in fact aid the algorithm's performance.

Pancake Puzzle

Figure 2-10 - 2-12 shows results on the pancake problem (Kleitman, Kramer, Conway, Bell, & Dweighter, 1975; Gates & Papadimitriou, 1979; Heydari & Sudborough, 1997) where the objective is to sort a sequence of pancakes through a minimal number of prefix reversals. We use the GAP heuristic (Helmert, 2010) for all the real-time search algorithms. We test three size of pancakes: 16, 32, and 40. One hundred instances of each size were tested per experiment.

Nancy outperforms LSS-LRTA*, in particular with small lookahead bounds. Furthermore, as we scale the instance size, the gap between Nancy and LSS-LRTA* becomes larger. While LSS-LRTA* performs better than Nancy with larger lookaheads on the 16-pancake instances, on the instances with 32 and 40 pancakes Nancy consistently finds better solutions.

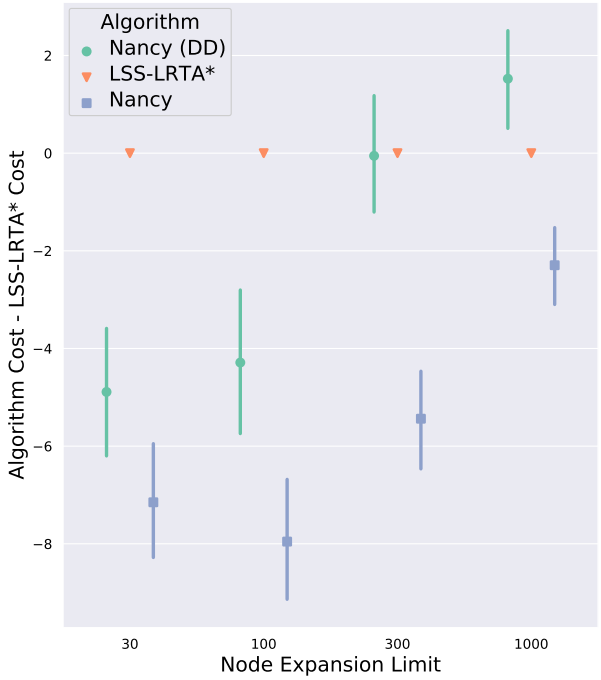


Figure 2-11: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 32 pancakes.

DDNancy inherits the better scaling from Nancy as the problems size is increased and outperforms LSS-LRTA* in particular for small lookahead limits, it does not match the performance of assumption-based Nancy. This is again due to the assumptions embodied in Nancy were likely heavily informed by search behavior, as we saw in Figure 2-5, the Gaussian assumption on the heuristic error is not seriously violated in this domain.

Racetrack

The Racetrack domain is very similar to the grid pathfinding problem, but features additional actions and inertia. It is reminiscent of autonomous driving and is a variant of the popular Racetrack problem (Barto, Bradtke, & Singh, 1995). Figure 2-13 shows the two maps used in our experiments. The blue cell is the initial location of the agent , the green cells are the goal locations, and black cells are static obstacles in the map. The track shown on the left was created by Hansen and Zilberstein (2001), and the cluttered track on the right by Cserna, Doyle, Ramsdell, and Ruml (2018). The agent moves in a grid attempting to reach one of a set of goal locations while avoiding static obstacles. Each action

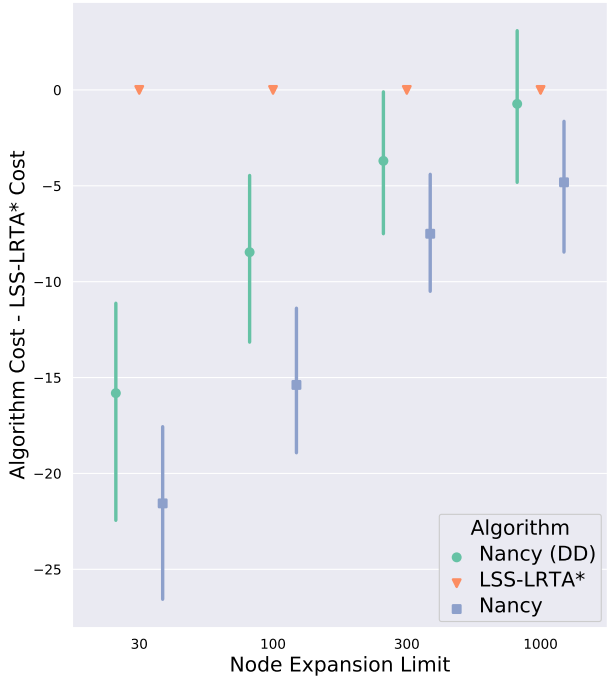


Figure 2-12: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the pancake puzzle with 40 pancakes.

modifies the acceleration of the agent by -1, 0, or 1 in both the horizontal and vertical directions, making for a total of 9 distinct actions. There is no limit on the agent's speed. The system state includes the agent's location and velocity. The objective is to minimize the number of time steps until a goal cell is reached. The heuristic function is the maximum, either horizontally or vertically, of the distance to the goal divided by an estimate of the maximum achievable velocity (MAV) in that dimension. MAV could possibly be attained given the domain size. We note the map width as w, and map height as h. The MAV value in the horizontal dimension, note as MAV_x, is the largest integer value such that $MAV_x(MAV_x + 1)/2 \leq w$ is still true. Similarly, MAV value in the vertical dimension, note as MAV_y, is the largest integer value such that $MAV_y(MAV_y + 1)/2 \leq h$ is still true. For each of the two maps, we created 25 instances with starting positions chosen randomly among those cells that were at least 90% of the maximum distance from a goal.

Figure 2-14 and 2-15 shows the results of Nancy and LSS-LRTA* on the two maps of Racetrack. On both maps Nancy significantly outperforms LSS-LRTA*. Like before, the difference is more pronounced for smaller lookahead bounds, and converge as the lookahead bound gets larger.

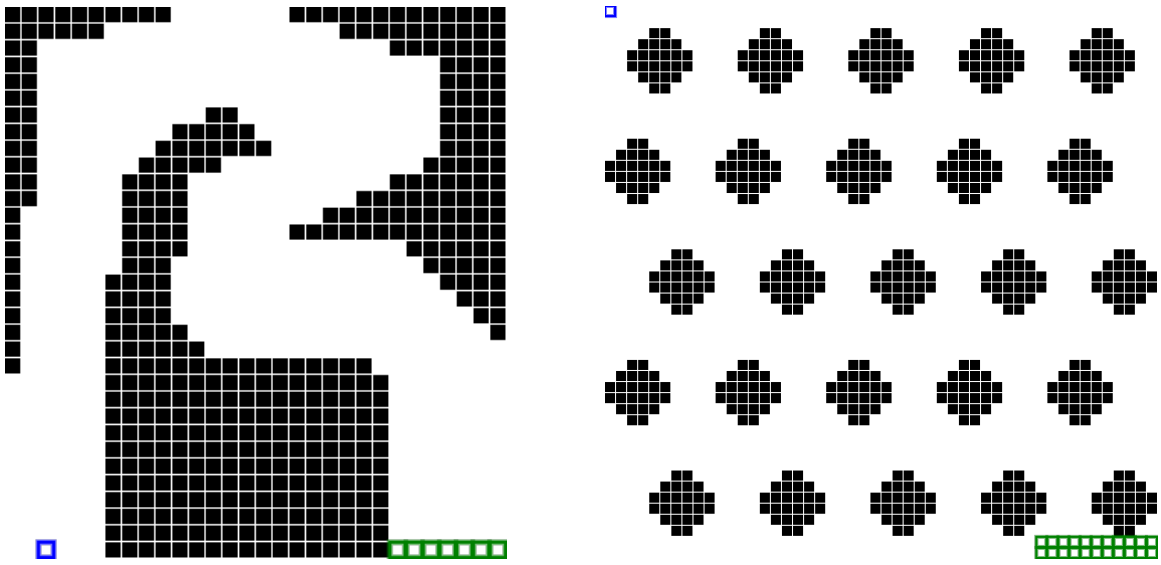


Figure 2-13: Barto (left) and uniform (right) Racetrack maps.

DDNancy has very similar performance to Nancy. While both beat LSS-LRTA*, DDNancy further improves the solution cost for the smallest tested lookahead bound (30 expansions).

Classical Planning

To evaluate our new algorithms on classical planning benchmark domains, Maximilian Fickert and Leonhard Staut (Fickert, Gu, Staut, Ruml, Hoffmann, & Patrik, 2020b) extended Fast Downward (Helmert, 2006) to facilitate real-time search algorithms bounded by a number of expansions and implemented LSS-LRTA* and the new Nancy variants in this framework. Since we need to gather data for each specific domain that we want to evaluate, we chose to focus our experiments on a few selected domains, Blocksworld, Transport, Elevators, where preliminary experiments showed assumption-based Nancy to be relatively ineffective compared to LSS-LRTA*. For Transport and Elevators, we include their unit-cost versions, and we omit the original-cost version of Elevators as it has zero-cost actions, making the considered algorithms incomplete. We ran the experiments on a cluster of Intel Xeon E5-2660 machines using the lab framework (Seipp, Pommerening, Sievers, & Helmert, 2017). We choose $h^{\text{LM-cut}}$ (Helmert & Domshlak, 2009) due to it being a popular admissible heuristic that achieves competitive performance without requiring the tuning of many parameters, and used the standard limits of 30 minutes and 4 GB memory in all experiments.

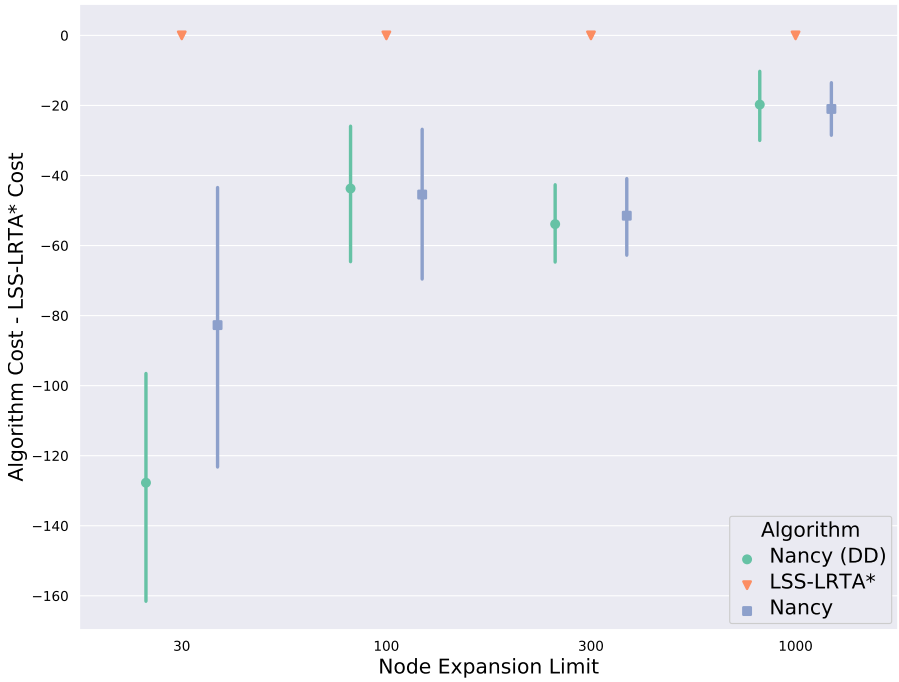


Figure 2-14: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the Barto map of the Racetrack domain.

The data for the data-driven variant was generated on the same number of instances (185) using the same size parameters as the IPC instances. On these instances we computed h^* for all states that were expanded by weighted A*. The set of h^*-values forms a distribution for each h-value which is looked up by DDNancy during search. Distributions for h-values that were not seen during training and can therefore not be looked up are extrapolated from the next lower h-value.

Table 2-1 shows the results of these experiments. We test lookaheads of size 100, 300, and 1,000 nodes. As a general trend, solution cost tends to decrease with larger lookahead for each algorithm, since a larger lookahead usually leads to more informed decisions. Comparing the individual algorithms with each other, we take LSS-LRTA* as the baseline. The original Nancy variant performs comparatively worse in almost all tested scenarios. The persistent assumption-based Nancy variant improves on the original Nancy algorithm across all domains, sometimes dramatically. Compared to LSS-LRTA*, however, it still performs poorly in Transport. The data-driven Nancy variant manages to perform well in all test cases. It stays competitive with all other algorithms and frequently has the lowest solution cost overall, even in transport where the assumption-based variant

Domain	Lookahead	LSS-LRTA*	Nancy	Nancy (DD)
Blocksw.	100	46	**33**	38
	300	36	**30**	34
	1000	30	32	**27**
Transport	100	631	615	**496**
	300	519	559	**485**
	1000	499	567	**422**
Transport (unit-cost)	100	48	40	**31**
	300	47	**30**	34
	1000	35	29	**27**
Elevators (unit-cost)	100	50	**35**	39
	300	32	**29**	30
	1000	34	27	**26**

Table 2-1: Geometric means of the solution cost on instances solved by all algorithms. The limit on the number of expanded nodes in the lookahead is denoted by L.

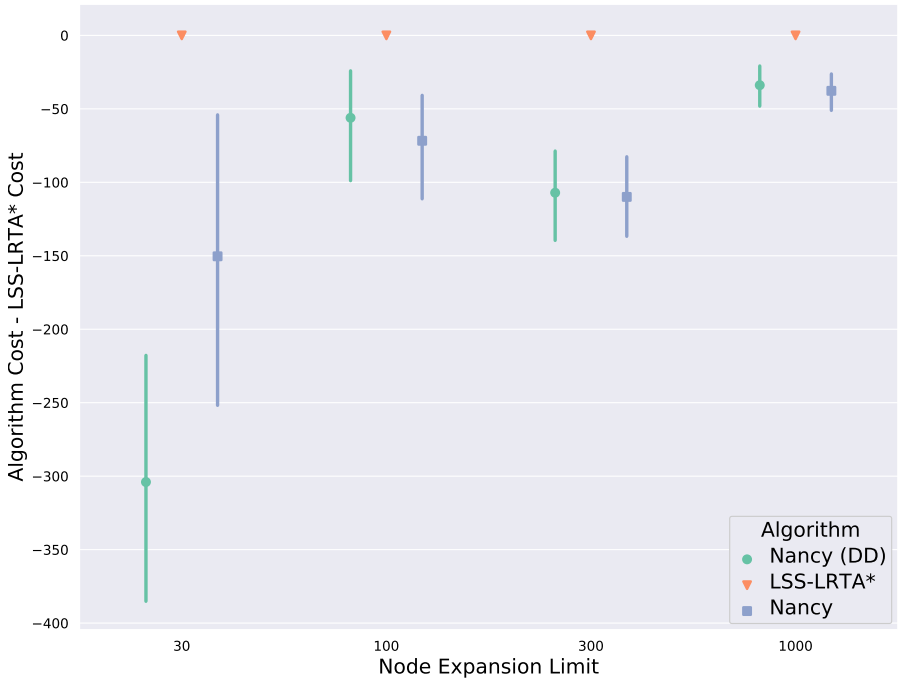


Figure 2-15: Solution cost relative to LSS-LRTA* with varying lookahead bounds on the uniform map of the Racetrack domain.

faltered.

Evaluation Under Time Bounds

Since Nancy's risk-based lookahead is more sophisticated than other expansion strategies, it is natural to ask if its added computational overhead prevents it from being useful in practice. To test this, we performed a comparison in which each planning iteration of the search algorithms was limited by time instead of number of expansions. While time bounds capture the setting of practical applications more faithfully, they also introduce technical challenges as the results may depend on implementation details (e.g. memory management), hardware details (e.g. caches), and other outside factors (like OS scheduling and other processes running in parallel), and make the results harder to reproduce. Furthermore, using strict time bounds would require the planner to immediately commit to an action once the bound is reached. This is a problem in the implementation, as the planner can not just perform lookahead until the bound is reached, but may realistically only check the bound in between expansions for example, and needs to perform the learning phase before starting the next

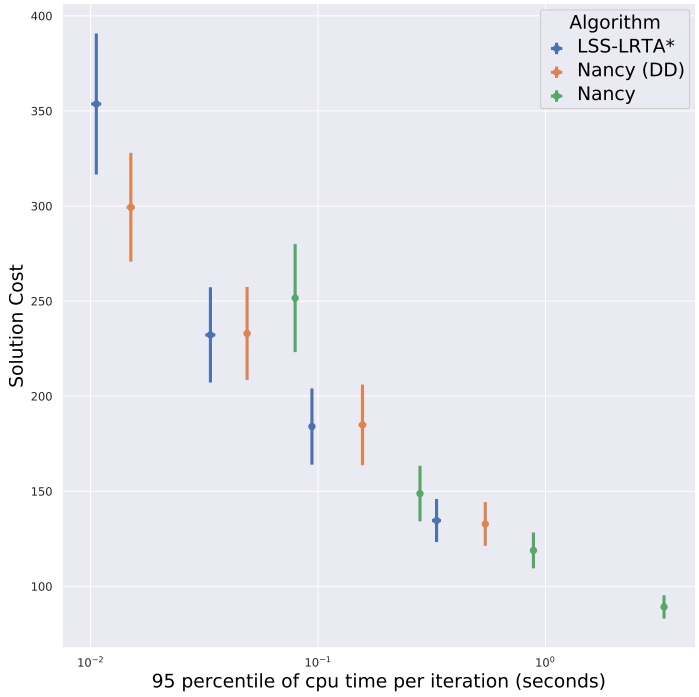


Figure 2-16: Solution cost as a function of CPU time on unit-cost 15-puzzle.

lookahead.

There are several ways to evaluate real-time search methods using time bounds (Cserna, 2019; Eifler, Fickert, Hoffmann, & Ruml, 2019). Here, we still run the experiments with expansion bounds, but we record the CPU time taken for each planning iteration. One way to compare algorithms would then be to compare the average solution cost achieved by each method versus the average time each took per planning iteration. Algorithms that achieve better solution cost for a similar time per iteration would be preferred, regardless of the number of expansions performed. However, average time per iteration would completely ignore variability in planning episode times. On the other hand, using the maximum time taken in any planning iteration makes the results very sensitive to sources of variability that are hard to control, such as memory management overhead. To balance these concerns, we used the 95th percentile of the time taken per planning episode.

Figure 2-16 - 2-22 shows the solution cost (y-axis) by 95th percentile CPU time among all iterations (x-axis). We show the mean with error bars indicating 95% confidence intervals on both x and y axis over all problem instances (100 for 15-puzzle and pancake, 25 for Racetrack). The mean and two error bars form a cross-style data point on the grid. Algorithms are plotted with different

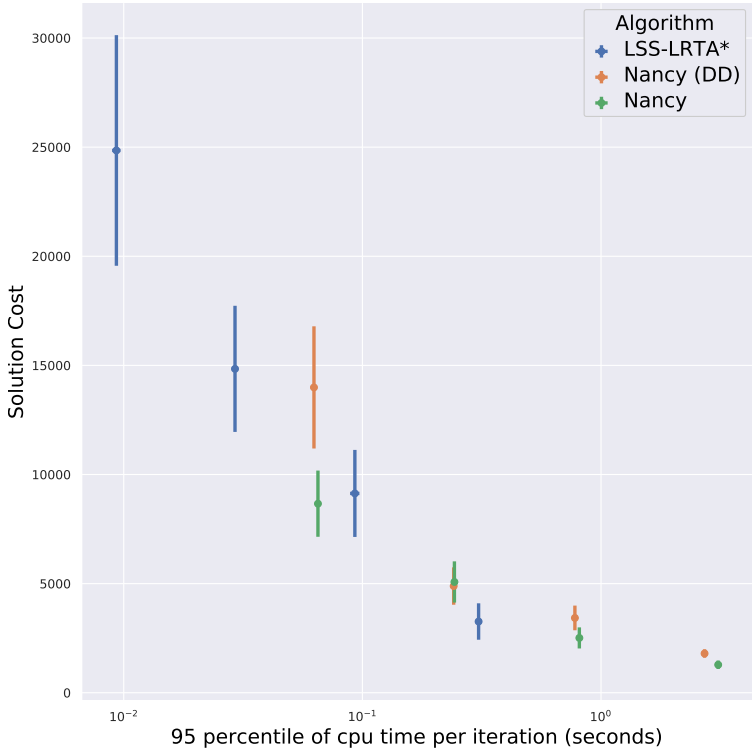


Figure 2-17: Solution cost as a function of CPU time on heavy-cost 15-puzzle.

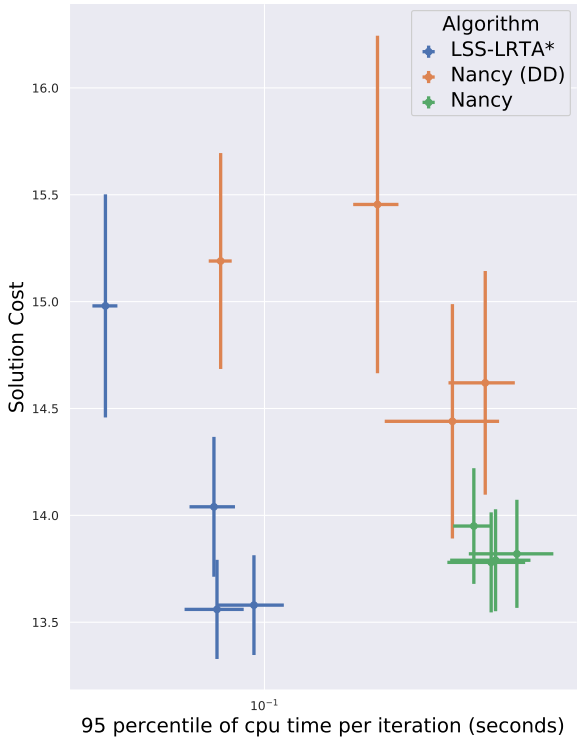


Figure 2-18: Solution cost as a function of CPU time on 16 pancake.

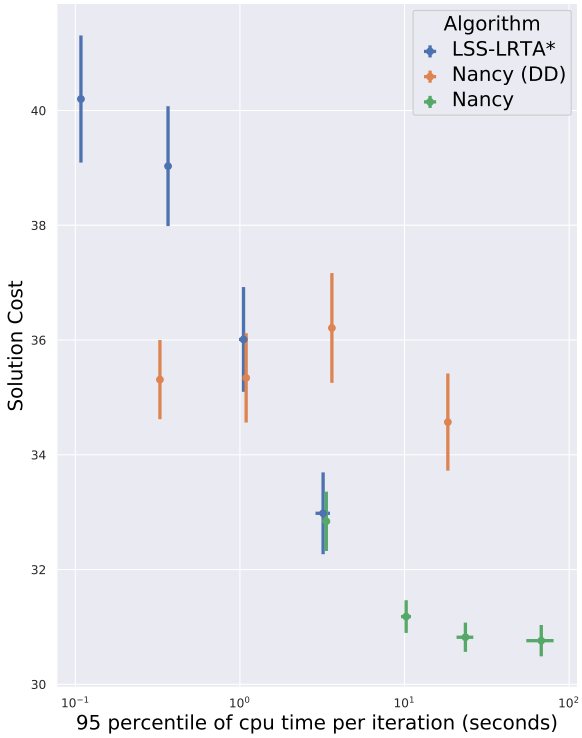


Figure 2-19: Solution cost as a function of CPU time on 32 pancake.

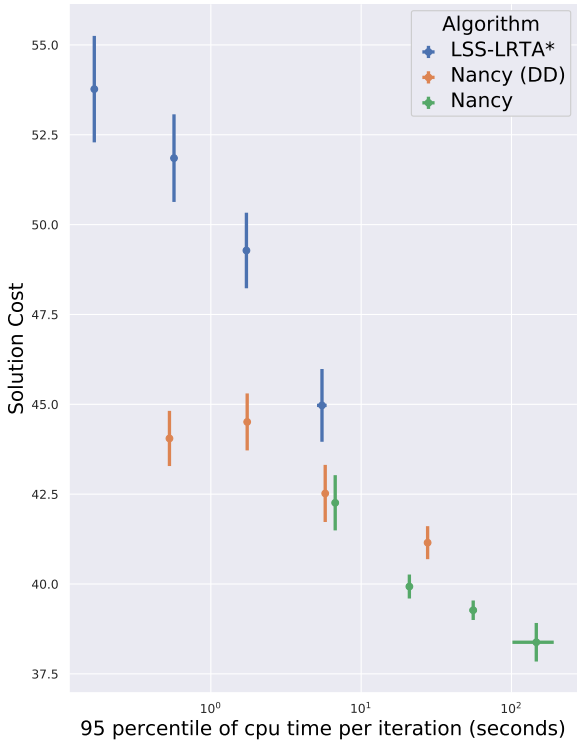


Figure 2-20: Solution cost as a function of CPU time on 40 pancake.

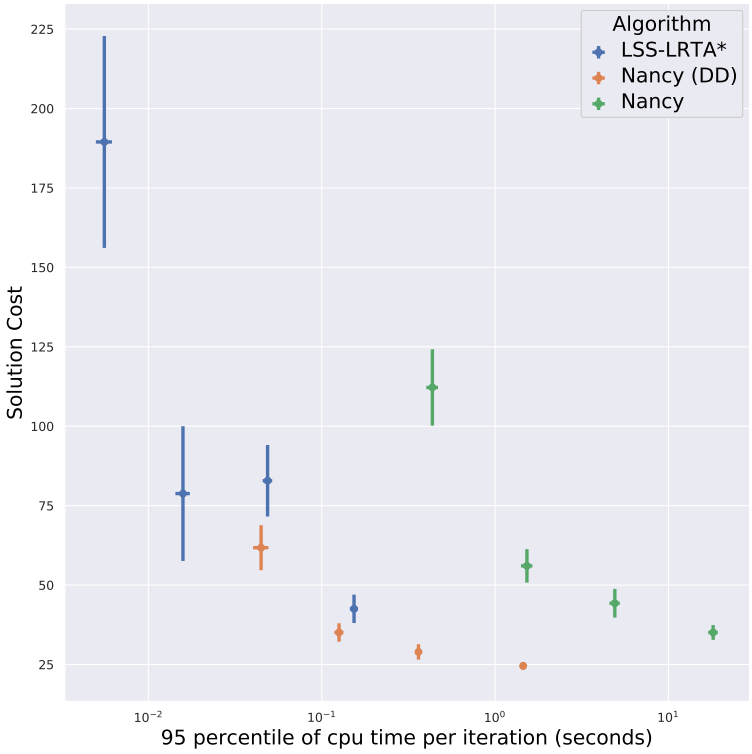


Figure 2-21: Solution cost as a function of CPU time on Barto map racetrack.

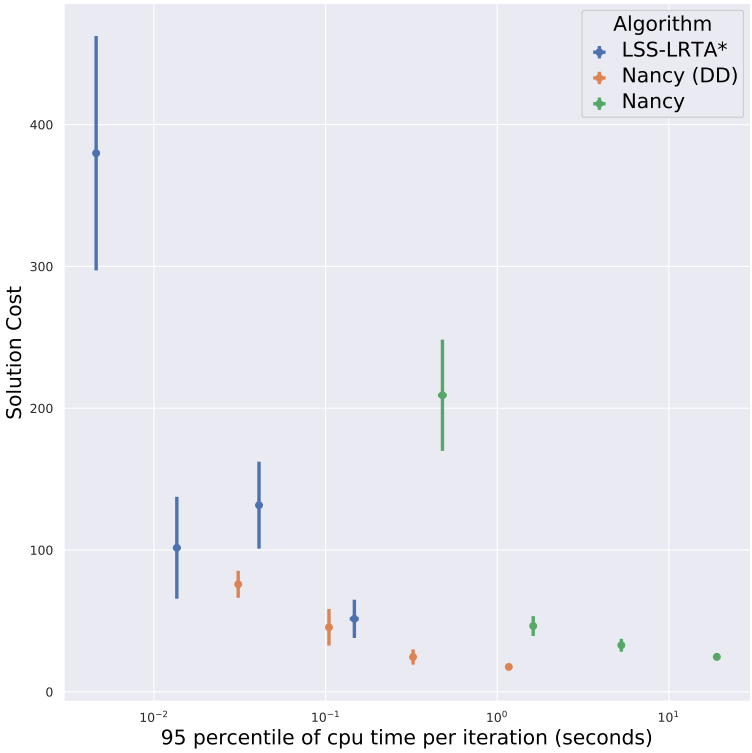


Figure 2-22: Solution cost as a function of CPU time in uniform map racetrack.

colors as indicated in the legend. For each algorithm, we consider expansion limits of 30, 100, 300, and 1000, from left to right.

The results confirm that Nancy is indeed slower than LSS-LTA* in terms of CPU time per node expansion. However, it takes Nancy many fewer expansions to achieve the same solution cost compared to LSS-LRTA*. For example, in the heavy-cost variant of the 15-puzzle (Figure 2-17), Nancy with just 30 expansions per lookahead (the left-most green data point) performs the same as LSS-LRTA* with a lookahead bound of 300 expansions (the third blue data point from the left). Thus, if we were to use a real time bound of 100 milliseconds, Nancy would indeed perform better than LSS-LRTA* on average, and such cases appear in other domains as well.

Compared to the assumption-based version, DDNancy has lower overhead per expansion. The main reason for this difference is that the assumption-based distributions are generated with more bins, so the risk calculation of DDNancy requires fewer iterations. Overall, when using time bounds, the observations are consistent with those for Nancy, and DDNancy also generally outperforms LSS-LRTA* in most cases.

In the pancake puzzle (Figure 2-18 - 2-20), Nancy gets better as the instance size is scaled up, consistent with the results when using expansion bounds. In the Racetrack domain (Figure 2-21 and 2-22), Nancy is outperformed by LSS-LRTA* when using time bounds. However, the data-driven instantiation of Nancy does perform well here.

2.4 Other Distributional Methods

While Nancy is, to our knowledge, the first method for real-time heuristic graph search that bases its search strategy on belief distributions, there has been previous work on other search methods that attempt to estimate and exploit value uncertainty. We consider two prominent approaches: interval estimation and Monte-Carlo tree search. In each case, we adapt previous work to our setting and empirically compare it to Nancy.

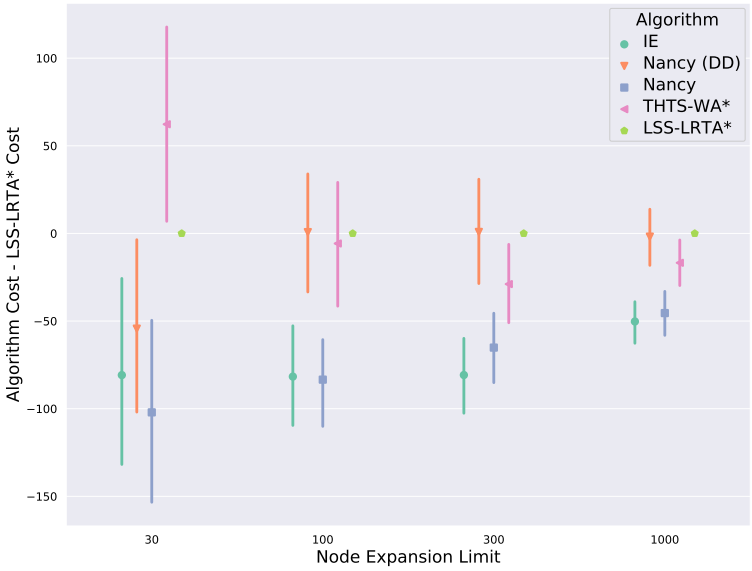


Figure 2-23: Comparison to IE and THTS on uniform-cost 15 puzzle.

2.4.1 Interval Estimation

Interval Estimation (IE) (Kaelbling, 1993; Strehl & Littman, 2004) is similar to Nancy, but when deciding under which which TLA the next should be expanded, IE chooses the TLA with the lowest lower bound on the 95% confidence interval of the backed up cost-to-goal estimate instead of performing a computationally complex risk analysis. In fact, empowered by the distribution, we can sort the open list under each TLA by the lower confidence bar and perform Nancy backups based on that. The interval estimation approach naturally practices the spirit of an exploration vs. exploitation balance in a very computationally efficient way.

Figure 2-23 - Figure 2-29 show experimental comparisons of Nancy and LSS-LRTA* to IE (and also to a Monte-Carlo tree search approach, which we discuss in the next subsection). IE performs well in our experiments, and closely matches the performance of Nancy. While Nancy has a slight advantage on the sliding tile and pancake puzzles, IE works marginally better on Racetrack. Overall, IE is very competitive, and yet simple to implement and computationally efficient.

Figure 2-24: Comparison to IE and THTS on heavy-cost 15 puzzle.

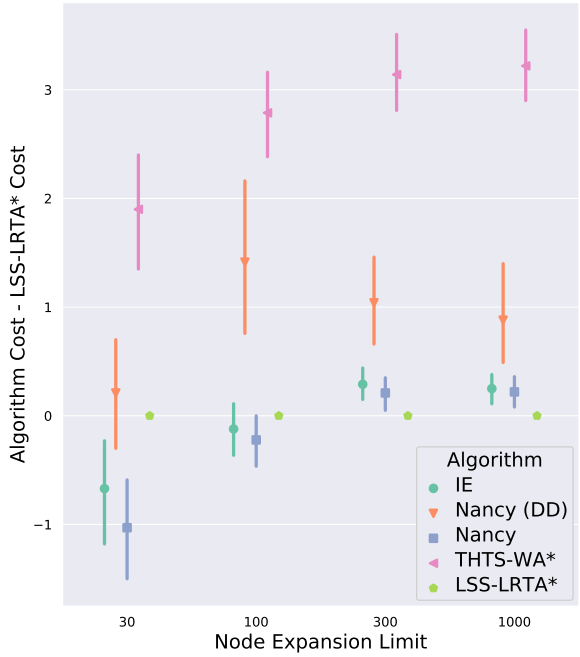


Figure 2-25: Comparison to IE and THTS on 16 pancake.

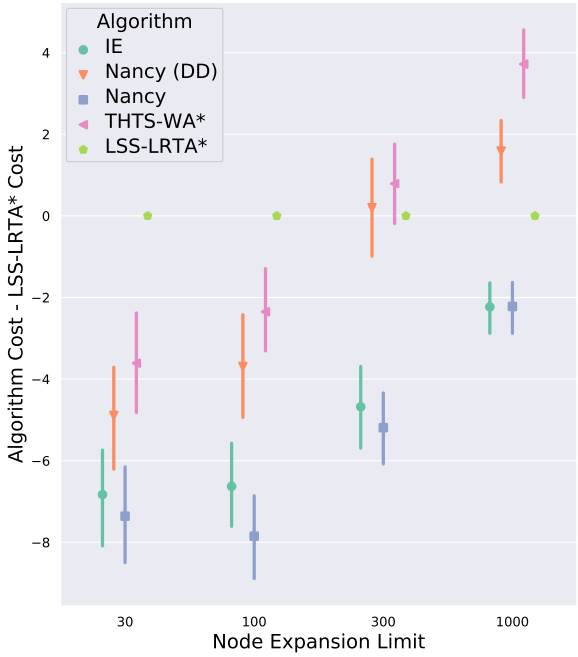


Figure 2-26: Comparison to IE and THTS on 32 pancake.

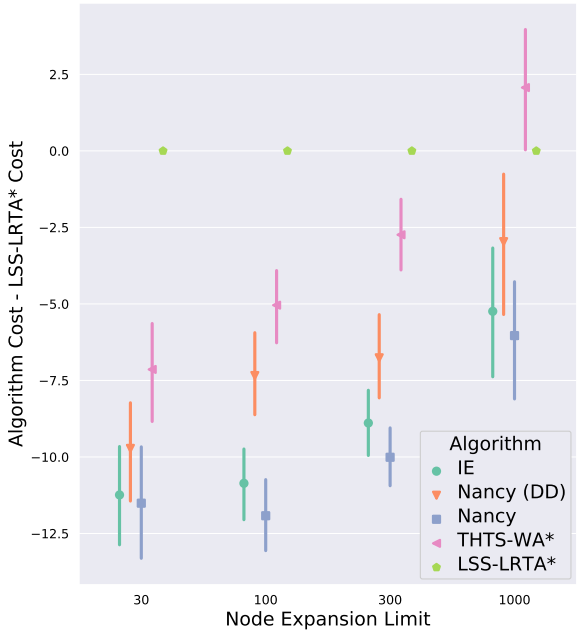


Figure 2-27: Comparison to IE and THTS on 40 pancake.

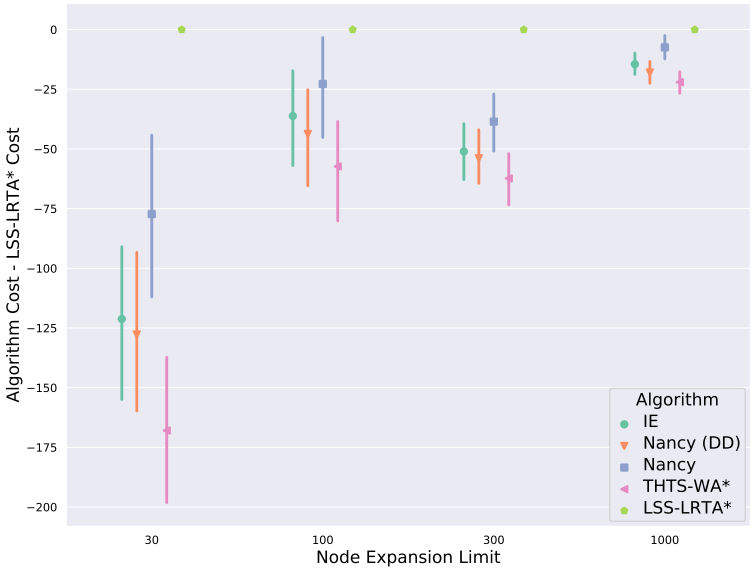


Figure 2-28: Comparison to IE and THTS on Barto map racetrack.

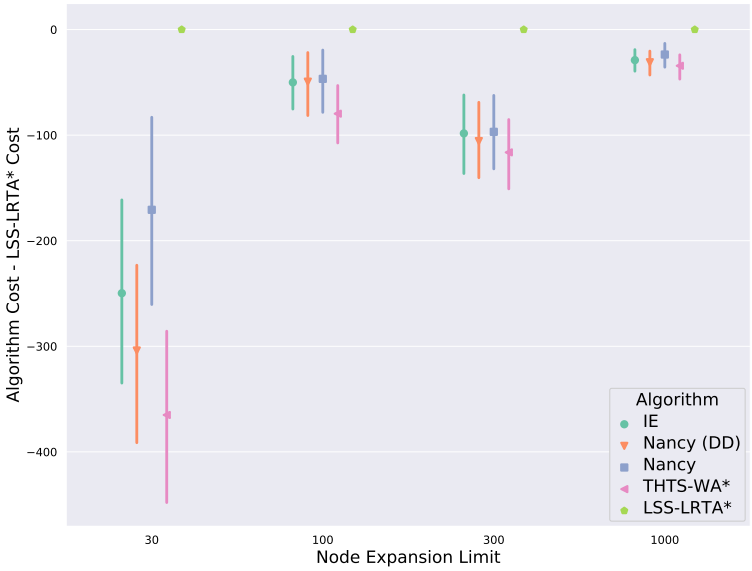


Figure 2-29: Comparison to IE and THTS on uniform map racetrack.

2.4.2 Monte Carlo Tree Search

Monte-Carlo tree search (Browne, Powley, Whitehouse, Lucas, Cowling, Rohlfshagen, Tavener, Perez, Samothrakis, & Colton, 2012) approaches such as UCT (Kocsis & Szepesvári, 2006) are popular for solving stochastic problems such as with MDPs (Keller & Eyerich, 2012) and POMDPs (Silver & Veness, 2010). Recently, Schulte and Keller (2014) adopted this to deterministic planning problems as trial-based heuristic tree search (THTS). Like Nancy, THTS also takes the uncertainty about the heuristic into consideration (though more implicitly), so we should compare it to our approach. However, THTS was only described as an offline search framework. We adapted it to the real-time setting based on LSS-LRTA*. We replace the A* lookahead in the expansion phase with the THTS algorithm (Schulte & Keller, 2014) . More precisely, we use the THTS-WA* instantiation of the THTS framework since this variation had the best results for the base setting in the original paper. In the learning phase, we also use a reversed Dijkstra's algorithm to update the heuristic values, working from the uninitialized frontier tree nodes inwards. In the decision making phase, we use the identical strategy as LSS-LRTA* and move towards the node with minimal f-value.

Consider again Figure 2-23 - Figure 2-29. The real-time variant of THTS performs poorly on the 15-puzzle (in particular the heavy-tile version). On the pancake puzzle, it again beaten by most of the other considered approaches, but it does beat LSS-LRTA* on the larger instances with small lookahead. On the Racetrack domain on the other hand, it outperforms all other algorithms, for all considered lookahead values.

2.4.3 Summary

We reported an experimental comparison with approaches from RL that exploit value uncertainty, such as Monte Carlo tree search and Kaelbling's interval estimation. We find that our approach, Nancy, generally outperforms previous methods, particularly on more difficult problems.

2.5 Conclusion

We reconsidered real-time search as a decision making process where the limited available information creates uncertainty. We studied a search method that makes use of novel strategies to model this uncertainty and reason about it to guide the search.

Our experimental results provide evidence that a real-time search can often improve its trajectory cost by using additional techniques when choosing where to do additional search. Acknowledging the uncertainty about the true distance to the goal, we propose techniques that model this uncertainty and reason about it explicitly. We have shown that this improves how the algorithm spends its time in the lookahead. With this observation, we can see how real-time heuristic search can benefit from representing uncertainty.

CHAPTER 3

Metareasoning for Bounded-Cost Search

Even if real-time reactivity is not required, many planning problems are too hard to solve optimally. In bounded-cost search, one attempts to find, as quickly as possible, a plan that costs no more than a user-provided absolute cost bound. In this chapter, we show how bounded-cost search can benefit from representing uncertainty. Several algorithms have been previously proposed for this setting, including Potential Search (PTS) and Bounded-cost Explicit Estimation Search (BEES). BEES attempts to improve on PTS by predicting whether nodes will lead to plans within the cost bound or not. This chapter introduces a relatively simple algorithm, Expected Effort Search (XES), which uses not just point estimates but belief distributions in order to estimate the probability that a node will lead to a plan within the bound. XES's expansion order minimizes expected search time in a simplified formal model. Experimental results on standard planning and search benchmarks show that it consistently exhibits strong performance, outperforming both PTS and BEES. We also derive improved variants of BEES that can exploit belief distributions.

This work appeared in the proceedings of IJCAI-21 (Fickert, Gu, & Ruml, 2021). Tianyi Gu implemented the algorithms in the domain-specific framework and conducted experiments in traditional search domains. Maximilian Fickert conducted experiments in the domain-independent framework and also conducted the optimal efficiency proof.

3.1 Introduction

In many real world applications, an optimal solution can be too hard to obtain within realistic constraints on CPU time and memory. In the field of suboptimal search, two main directions have been pursued. The first direction focuses on producing bounded suboptimal solutions that are within a given factor w of the optimal cost. The most famous algorithm in this group is weighted A^*

(Pohl, 1970). The second direction, called bounded-cost search (Stern et al., 2011; Haslum, 2013), considers an alternative setting by providing an absolute cost bound C. The search algorithms are then designed to find a plan with cost less than or equal to C as quickly as possible. Bounded-cost search corresponds to many realistic cost-constrained scenarios such as planning under a fixed budget for plan execution. In this chapter, we focus on the second direction and design a new bounded-cost search algorithm called Expected Effort Search (XES).

For bounded-cost search, one can take advantage of the problem setting and design heuristics that can guide the search to find solutions within the given cost bound as quickly as possible. For example, potential search (PTS) (Stern et al., 2011) attempts to expand the node with the maximum probability of leading to a plan within the bound. However, PTS does not consider the search effort that would be required to find a goal under a search node. Bounded-cost Explicit Estimation Search (BEES) (Thayer et al., 2012) attempts to improve upon PTS by incorporating distance-estimating heuristics. BEES attempts to expand the node, among those estimated to be within the bound, that is closest to a goal. However, BEES does not take the uncertainty of its estimate into account. A node with an expected cost estimate barely below the cost bound is considered equally important as a node with expected cost far less than the cost bound.

In this chapter, we propose a new bounded-cost search algorithm called Expected Effort Search (XES), which takes both kinds of information into account. Like BEES, XES uses distance-to-go and inadmissible cost estimates, but it also uses belief distributions to model their uncertainty. XES estimates two quantities of a node n: 1) Estimated search effort $T(n)$, and 2) the probability $p(n)$ of finding a solution within the cost bound. It then performs best-first search on $T(n)/p(n)$. This idea was first proposed by Dobson and Haslum (Dobson & Haslum, 2017) and implemented as an internal objective function for specific domain-independent planning heuristics. In this chapter, we extend it to the general heuristic search setting, showing how such estimates can be obtained for arbitrary heuristics. Our experimental results show that XES consistently outperforms previous bounded-cost search methods. Furthermore, we devise new variants of BEES that make use of the belief distributions used in XES, which improves performance on our benchmark set.

3.2 Previous Work

A bounded-cost search problem can be defined as a six-tuple $\langle S, s_{init}, succ(s), c(s,s'), G, C\rangle$, where S is a set of states, $s_{init} \in S$ is the initial state, $succ(s)$ is the transition function yielding the set of successor states of s, $c(s,s')$ is the cost function of the transition from state s to its successor $s' \in succ(s)$, $G \subseteq S$ is the set of goal states, and C is a given constant. The task is to find a path from s_{init} to a goal state (a *plan*) whose sum of transition costs is less than or equal to C.

Other than the notation specified in chapter 1, we use $d(n)$ to represent an estimate of the number of state transitions required to reach a goal (i.e., distance-to-go) from a search node n. The estimate d is often as easy to compute as h just by counting the number of actions in the solution to a relaxation of the original problem.

One simple strategy for bounded-cost search is to prune all the nodes with $g(n) > C$ (or $f(n) > C$ if h is admissible) during a (weighted) A* or greedy search. However, this may waste work as the heuristic that guides node selection is insensitive to the bound. This has motivated the development of several specialized algorithms for the bounded-cost setting.

3.2.1 Potential Search

Stern et al. (Stern et al., 2011) define the potential of a node as the probability that the node will be part of a solution whose cost is within the user-provided cost bound C: $P^C(n) = Pr(h^*(n) \leq C - g(n))$. PTS is a best-first search on $P^C(n)$ and thus guides the search toward nodes which have high probability of finding a solution within the cost bound. However, instead of explicitly calculating $P^C(n)$, which is not trivial to do, PTS constructs a function $f_{lnr}(n) = \frac{h(n)}{C-g(n)}$. Stern et al. (Stern et al., 2011) show that $f_{lnr}(n)$ yields the same ordering as $P^C(n)$ under the assumption that the error between the true cost-to-go $h^*(n)$ and its estimate $h(n)$ is linear in the size of $h(n)$. Thayer et al. (Thayer et al., 2012) show that PTS can be enhanced by an inadmissible heuristic $\hat{h}(n)$ when one is available. Let $\widehat{\text{PTS}}$ denote a best-first search on $\hat{f}_{lnr}(n) = \frac{\hat{h}(n)}{C-g(n)}$.

The limitation of potential-based approaches is that they only consider the probability that a node is on a path to a goal under the bound and yet never take into account search effort, i.e. the time needed to find that solution. Potential on its own does not include a term encouraging the search to

reach a goal. Since the task of bounded-cost search is to find a plan within the bound as quickly as possible, search effort estimates ought to be useful for search guidance.

3.2.2 Bounded-cost Explicit Estimation Search

Building on Thayer and Ruml's [(2011)] EES algorithm for bounded suboptimal search, Thayer et al. (Thayer et al., 2012) introduced Bounded-cost Explicit Estimation Search (BEES) and Bounded-cost Explicit Estimation Potential Search (BEEPS). BEES uses three sources of estimates to guide the search: an admissible cost heuristic h, an inadmissible cost heuristic $\hat{h}$, and an inadmissible distance-to-go heuristic $\hat{d}$. In addition to the standard open list, $open^C$, which contains all nodes with $f(n) = g(n) + h(n) \leq C$ sorted by f, BEES also maintains a focal list, $\widehat{open}^C$, which contains all the nodes whose inadmissible estimate is within the cost bound, i.e., $\hat{f} = g(n) + \hat{h}(n) \leq C$, sorted by $\hat{d}$. Nodes in $\widehat{open}^C$ are expanded first, falling back on $open^C$ only if $\widehat{open}^C$ is empty. BEEPS is a variation of BEES, differing only in ordering the standard open list, $open^C$, on $\hat{f}_{lnr}(n)$.

3.2.3 Combining Cost and Distance Estimates

Dobson and Haslum (Dobson & Haslum, 2017) attempt to improve BEEPS by merging cost and distance heuristics into a single combined heuristic rather than alternating between them using two queues. They point out that the probability that a node n will be part of a solution whose cost is within the bound C, $P^C(n)$, indicates that we might expect to expand a total of $\frac{1}{P^C(n)}$ many n-like nodes whose subtrees are similar to n, before a solution is found. Given this observation, they design an *expected work* heuristic $H^C(n) = T^C(n)/P^C(n)$, where $T^C(n)$ is the prediction of the size of the C-bounded subtree rooted at n. They describe how pattern database heuristics (Culberson & Schaeffer, 1996) can be adapted to compute H^C by modifying the internal objective function for the abstract plans, and show that it performs well on selected planning domains.

Dobson and Haslum (Dobson & Haslum, 2017) approximate $P^C(n)$ with the potential value used in PTS (i.e, $f_{lnr}(n)$) and estimate $T^C(n)$ with distance-to-go. They point out the potential value could be a poor approximation for the probability and they speculatively propose that one could perhaps approximate $P^C(n)$ via online learning of heuristic errors. In this chapter, we further

study this idea, show how it can be implemented for arbitrary underlying heuristics, and compare it to the existing state-of-the-art techniques for bounded-cost search.

3.2.4 Search Using Belief Distributions

In chapter 2, it has been shown that a heuristic search can benefit from being guided by belief distribution of cost estimates To recall, Nancy (Mitchell et al., 2019) is a real-time search algorithm that uses distributions of cost-to-go estimates to choose its next action. The distributions are derived from online observations of the one-step error of the heuristic and distance (Thayer, Dionne, & Ruml, 2011): for a search node n, Nancy creates a Gaussian distribution $N(\hat{f}(n), (\frac{\hat{f}(n)-f(n)}{2})^2)$ that is centered around the debiased total cost estimate $\hat{f}(n)$ and estimates the standard deviation as half of the difference between $\hat{f}(n)$ and $f(n)$.

3.3 Expected Effort Search

Extending the line of work initiated by Dobson and Haslum (Dobson & Haslum, 2017), we propose a new bounded-cost search algorithm called Expected Effort Search (XES). We design a combined expected effort heuristic that accounts for both the probability of finding a solution within the cost bound and the effort required to do so. We first summarise a simplified formal model, given by Maximilian Fickert (Fickert et al., 2021), to give a theoretical justification for this heuristic and then show how the probability estimate of finding a solution within the bound can be instantiated based on the belief distributions used by the Nancy algorithm (Mitchell et al., 2019).

3.3.1 A Simple Formal Model

Let $p(n)$ be an estimate of a node n's potential, i.e., the probability that there is a solution under n within the cost bound. In their model, Fickert et al. (2021) uses three assumptions. The first assumption (A1) is that the search procedure works exclusively in one subtree at a time. Let $T(n)$ be an estimate of the search effort to find a solution under n. The search effort can be defined as how much search (i.e., expansions) has to be done before find a solution. One can try to predict the size of the subtree under n to estimate the remaining search effort of a node n. In the experiment section

below, we use distance-to-go to predict the search effort under the assumption that the subtree size is proportional to the distance-to-go. A better assumption could be having subtree size growing exponentially as the distance-to-go grows. However, using an exponential function to predict the search effort could introduce an extra algorithm parameter for the exponent, which we leave as future work. Performing search below a node n can have two possible outcomes: either (a) a solution will be found under n with expected effort $T(n)$, or (b) the search does not find a solution under n. The second assumption (A2) is that the subtrees are independent: not finding a solution in one subtree does not affect the probability of finding a solution in any other subtree. The third assumption (A3) is that, for case (b), they assume that the search abandons that subtree after having spent $T(n)$ time, the same as in case (a).

Now, let $\sigma = \langle n_0, n_1, \ldots, n_m \rangle$ be the ordering of the search nodes that are currently in the open list. First case is where a solution is found below n_0, spending $T(n_0)$ time overall with a probability of $p(n_0)$. Otherwise, we need to next explore the search tree below n_1, where we would find a solution with overall probability $(1 - p(n_0))p(n_1)$ (considering the remaining probability of not finding a solution below n_0) and expected overall search time $(T(n_0) + T(n_1))$ since we explore both the search trees below n_0 and n_1, and so on. Abusing notation, let $T(n + m) = T(n) + T(m)$. Then, the expected overall search time for this ordering is

$$\begin{aligned} E(\sigma) =& p(n_0)T(n_0) + \\ & (1 - p(n_0))p(n_1)(T(n_0 + n_1)) + \\ & (1 - p(n_0))(1 - p(n_1))p(n_2)(T(n_0 + n_1 + n_2)) + \\ & \ldots \end{aligned}$$

We note that there will be a final term in this expression representing the time in the case in which there is no solution; this quantity might be finite or infinite depending on the search tree and search procedure.

If we want to know whether we should prefer a node n over another node m, we can, without loss of generality, compare the open list orderings $\sigma_n = \langle n, m, \ldots \rangle$ and $\sigma_m = \langle m, n, \ldots \rangle$. Note that all but the first two terms of the corresponding expected search times are identical. We will use this

to show that for ordering the open list, it is sufficient to use this shortened expression for expected search effort:

$$xe(n) = T(n)/p(n)$$

Theorem 2 *Given two open list orderings* $\sigma_n = \langle n, m, \ldots \rangle$ *and* $\sigma_m = \langle m, n, \ldots \rangle$, $E(\sigma_n) < E(\sigma_m) \Leftrightarrow xe(n) < xe(m)$.

Proof: We start with $E(\sigma_n) < E(\sigma_m)$. Removing equal terms from both sides yields

$$p(n)T(n) + (1 - p(n))p(m)(T(n+m)) <$$
$$p(m)T(m) + (1 - p(m))p(n)(T(m+n)).$$

Multiplying out, we get

$$p(n)T(n) + p(m)(T(n+m)) - p(n)p(m)(T(n+m)) <$$
$$p(m)T(m) + p(n)(T(m+n)) - p(m)p(n)(T(m+n)).$$

Simplifying by removing equal terms yields

$$p(n)T(n) + p(m)(T(n+m)) <$$
$$p(m)T(m) + p(n)(T(m+n)).$$

Then by multiplying out again we get

$$p(n)T(n) + p(m)T(n) + p(m)T(m) <$$
$$p(m)T(m) + p(n)T(m) + p(n)T(n)$$

and after removing equal terms again:

$$p(m)T(n) < p(n)T(m).$$

Finally, dividing by $p(n)$ and $p(m)$ yields

$$T(n)/p(n) < T(m)/p(m).$$

□

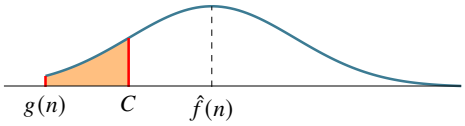


Figure 3-1: Estimating $p(n)$.

3.3.2 Estimating Expected Search Effort

XES is a best-first search on the expected search effort $xe(n) = T(n)/p(n)$. The key question is how to obtain the estimates $T(n)$ and $p(n)$. In this work, we instantiate $T(n)$ with the debiased distance estimate $\hat{d}(n) = \frac{d(n)}{1-\epsilon_d}$, where ϵ_d is the mean one-step error in d (Thayer et al., 2011), and we derive $p(n)$ from Nancy-style probability distributions. For a search node n, we use the same Gaussian distribution as Nancy, but truncated at a lower bound of $g(n)$ ($f(n)$ if h is admissible). The probability of finding a solution under n within the cost bound C is then just the area below the distribution up to C, as illustrated in Figure 3-1. This can be calculated directly using the cumulative density function for truncated Gaussian distributions (Hald, 1952).

XES weighs the potential of a node against its estimated search effort. If the cost bound is very generous, then $p(n)$ should be close to 1 for most nodes and XES converges to a best-first search on $\hat{d}$, effectively guiding the search to a goal as quickly as possible. We prune nodes with $h(n) = \infty$ (dead ends) and nodes with $g(n) > C$, but not nodes with $T(n)/p(n) = \infty$ as this case can occur when the heuristic overestimates significantly (so $p(n)$ is very low and may round to zero). XES satisfies the usual completeness guarantee from a best-first search, as all nodes that may satisfy the cost bound are eventually expanded until a solution is found.

3.4 BEES with Explicit Probability Estimates

BEES and BEEPS predict whether a node will lead to a solution within the cost bound using the criterion $\hat{f}(n) \leq C$, in which case n is inserted into the prioritized focal list instead of the standard open list. The explicit probability estimates derived from the Nancy-style belief distributions allow us to introduce new variants of these algorithms, BEES95 and BEEPS95, where we instead check if the probability is sufficiently high: $p(n) \geq 0.95$.

3.5 Experimental Evaluation

Although XES would appear to be appealing when the simplifying assumptions A1–A3 hold, it remains to be seen how it performs in practice. We empirically evaluate XES and the new BEES variants on both planning and heuristic search benchmarks. We follow Thayer et al. (Thayer et al., 2012) and implement PTS with $p^C(n) = \frac{h(n)}{1-g(n)/C}$ as opposed to $\frac{h(n)}{C-g(n)}$, which not only makes it clearer that PTS and $\widehat{\text{PTS}}$ converge to GBFS for large cost bounds, but also avoids precision issues for large C values. To compute the debiased heuristic $\hat{h}(n)$ for $\widehat{\text{PTS}}$, BEES, and XES, we use $\hat{h}(n) = h(n) + \bar{\epsilon_h} \cdot \hat{d}(n)$ where $\bar{\epsilon_h}$ is the mean one-step error in h (Thayer et al., 2011). We initialize ϵ_h with 100 virtual samples to avoid a large fluctuation of $\hat{h}$ values at the beginning of the search. The initial value is set to make $\hat{h}$ optimistic, using an initial one-step error of zero on the search domains (where the heuristics are admissible), and -0.5 on the planning domains (where the heuristic is inadmissible).

3.5.1 Search Domains

We empirically compare XES and BEES95 to PTS, $\widehat{\text{PTS}}$, and BEES on four classic heuristic search benchmarks. We do not include A*, GBFS, speedy search, or BEEPS because Thayer et al. (Thayer et al., 2012) found them to be dominated by BEES and $\widehat{\text{PTS}}$ Domain-specific solvers were implemented in C++ and run on 64-bit Linux systems with 3.16 GHz Intel E8500 Core2 duo processors and 8 GB of RAM. Algorithms were cut off at 7 GB RAM use. Beforehand, we solve each problem instance optimally, recording the cost. Then, in the tests, we set the cost bound of each instance to a factor of the optimal solution cost.

We show the results using four measures of performance: number of nodes generated, CPU time, coverage, and par10. We show them each as a function of the cost bound: the x-axis shows relative cost bound as a factor of optimal (e.g, a value 2 means a cost bound of twice the optimal solution cost is used for each instance). The y-axis shows the performance measure on a log scale. For number of nodes generated and CPU time, the line plots the mean across the commonly solved instances among all algorithms for all the bounds, with error bars indicating 95% confidence intervals. The number of instances used is shown in the legend. The coverage plots show the number of total solved instance

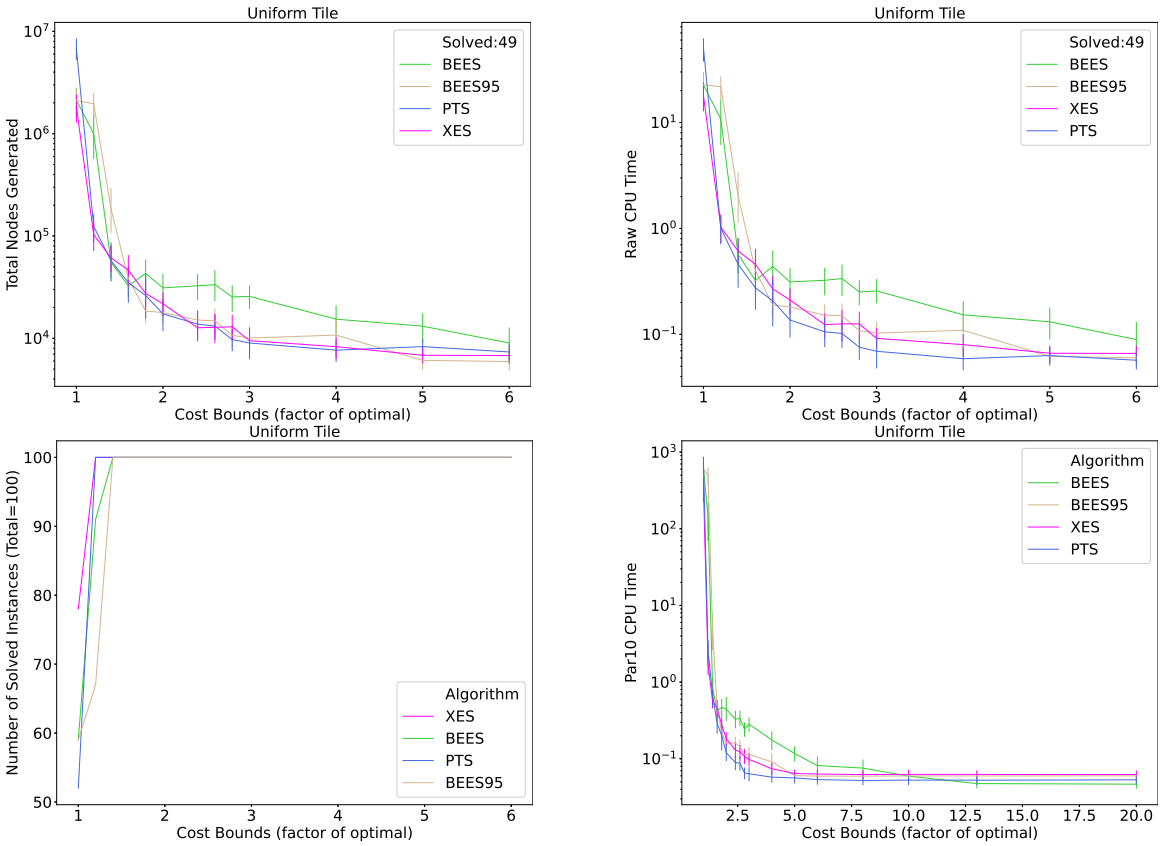


Figure 3-2: Performance measurements as a function of the cost bound on uniform tile.

for each algorithm. The par10 plots show the adjusted CPU time by treating each unsolved instance as solved but using 10 times the CPU time of the slowest instance among all solved instances.

Sliding-Tile Puzzle. We consider four variants of the well-known 15-puzzle: uniform cost, heavy cost (cost=tile#), inverse cost (cost=1/#), and square-root cost (cost=$\sqrt{\#}$). We use the Manhattan Distance as the heuristic h, weighting the components as appropriate (Thayer & Ruml, 2011), using the classic 100 start states published by Korf (Korf, 1985).

In Figure 3-2, 3-3 and 3-4, we show the results for the tile puzzle. We sort the variants by their difficulty from start (Figure 3-2) to end (Figure 3-4), with the easiest uniform-cost tile at beginning. Looking a the most important measurement, CPU time, as the problems get harder, XES BEES95 and $\widehat{\text{PTS}}$ start to outperform BEES and PTS. To have a reasonable number of commonly solved instances, we exclude PTS in the inverse cost plot due to its low coverage for tight bounds.

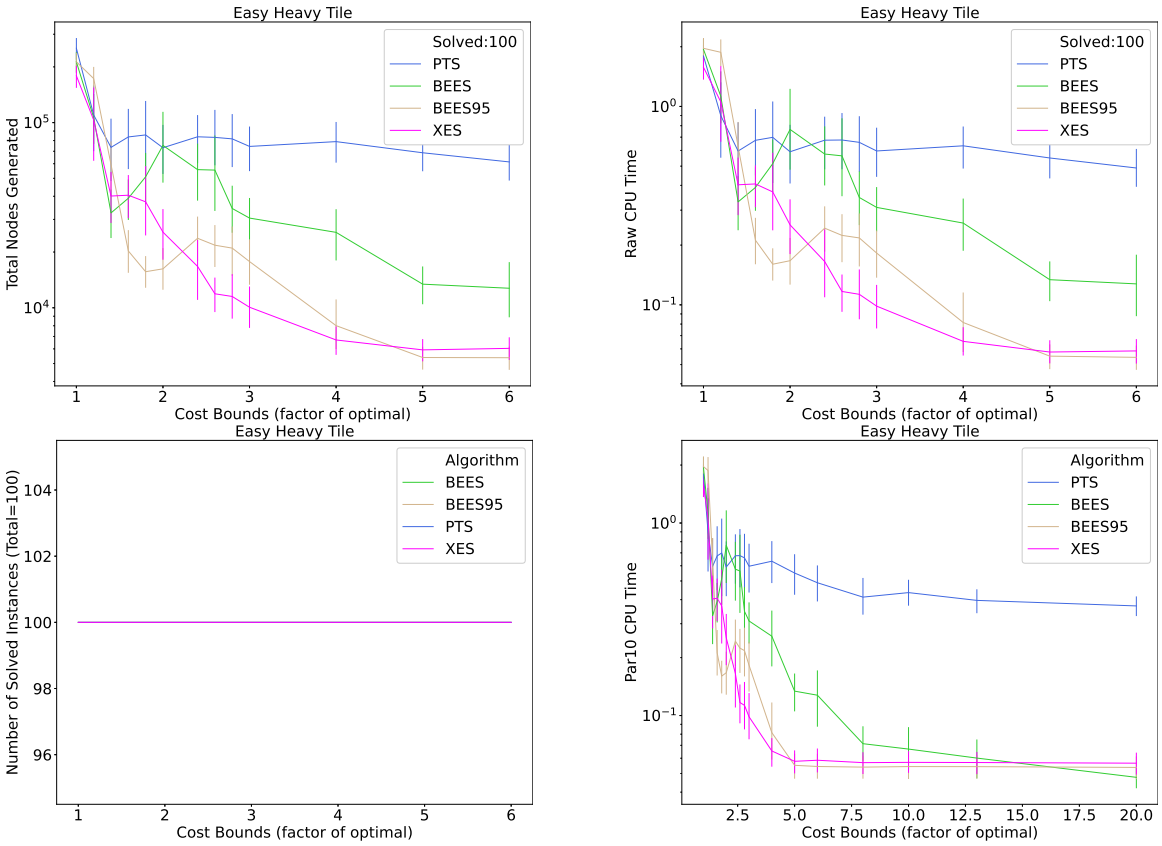


Figure 3-3: Performance measurements as a function of the cost bound on heavy tile.

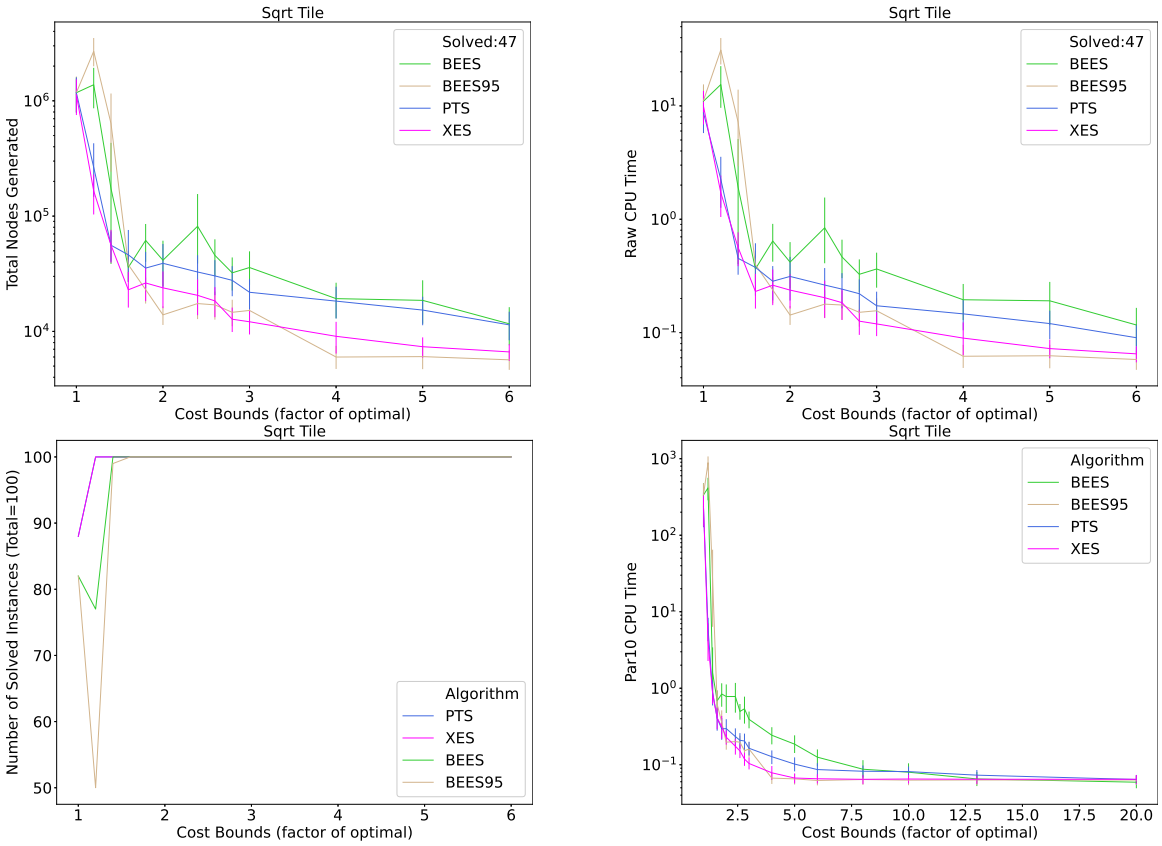


Figure 3-4: Performance measurements as a function of the cost bound on square-root tile.

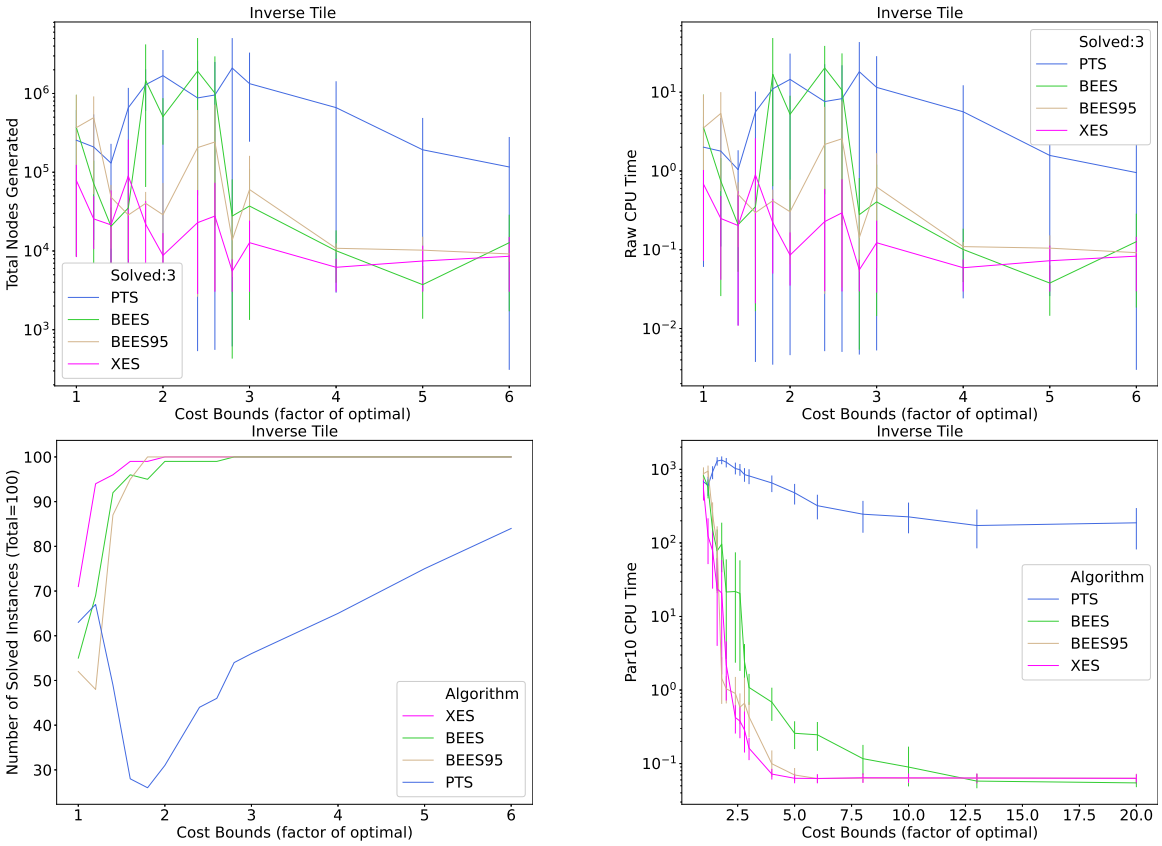


Figure 3-5: Performance measurements as a function of the cost bound on inverse tile.

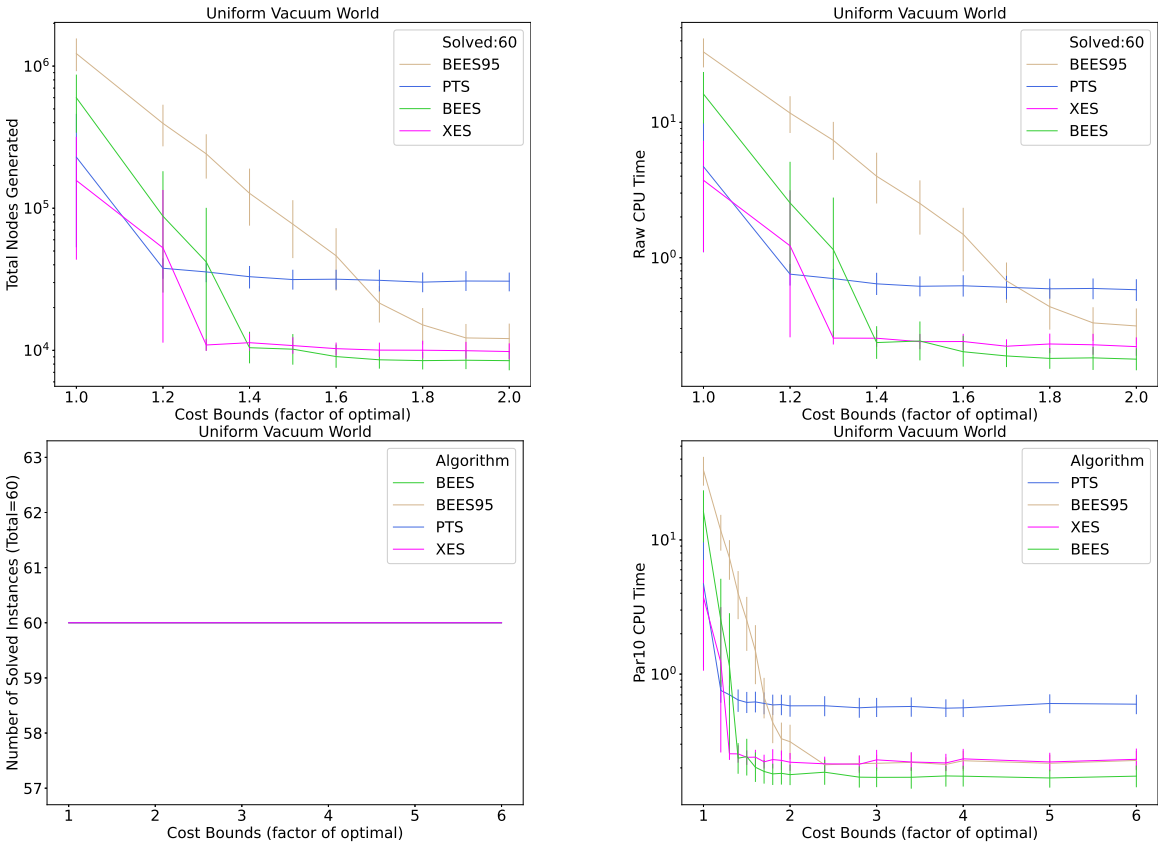


Figure 3-6: Performance measurements as a function of the cost bound on uniform vacuum world.

Vacuum World. The vacuum world domain was introduced by Thayer and Ruml (Thayer & Ruml, 2011), following the first state space presented in Russell and Norvig's textbook (1995) . We use 60 solvable instances of size 200×200, each cell having a 35% probability of being blocked. We again consider two variants with uniform and heavy costs, using the minimum spanning tree heuristic for h on the uniform-cost version and an adaptation thereof for the heavy-cost variant (Thayer & Ruml, 2011). The robot and the dirt piles are randomly placed in unblocked cells (10 piles for unit, 6 for heavy).

In Figure 3-6, we show the results of the uniform vacuum problem. XES behaves competitive to BEES, PTS and $\widehat{\text{PTS}}$. BEES95 performs poorly, especially when the bounds are tight. In heavy vacuum (Figure 3-7), XES and BEES95 are very close to BEES and outperform $\widehat{\text{PTS}}$ and PTS by one and two orders of magnitude respectively.

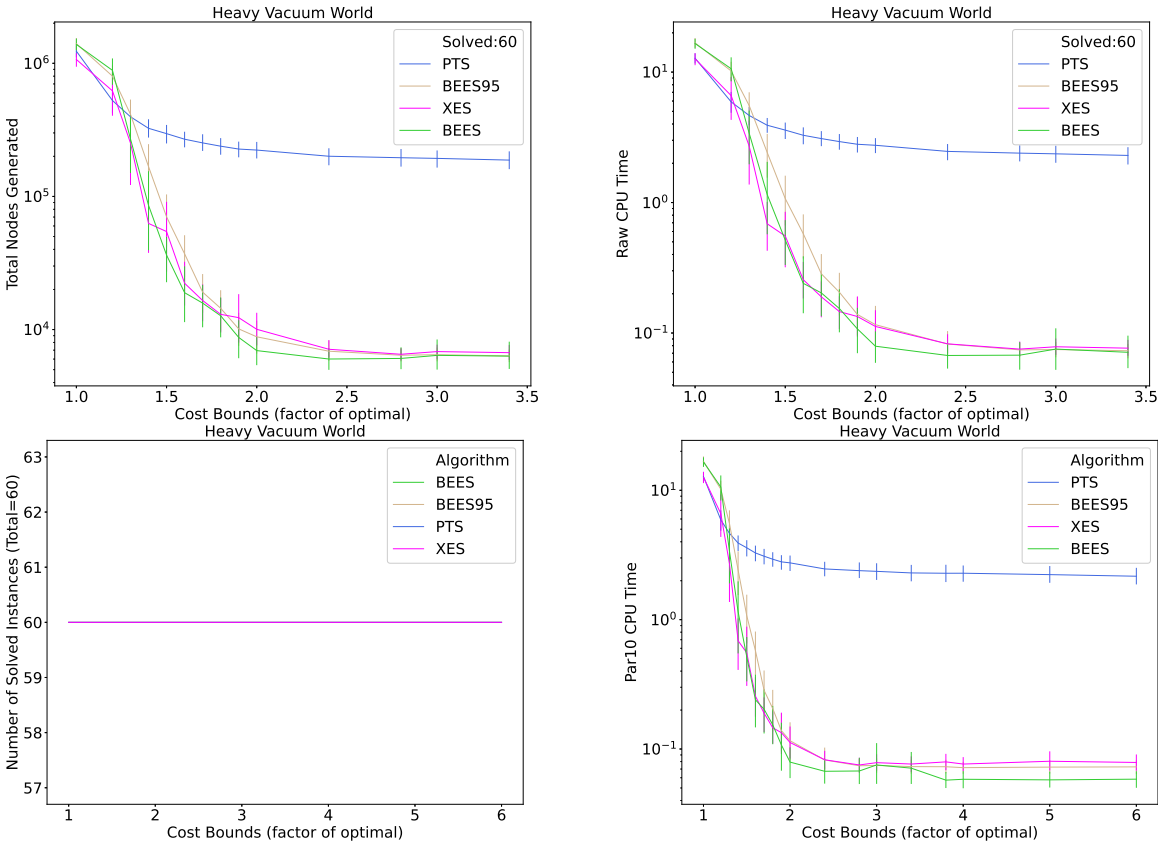


Figure 3-7: Performance measurements as a function of the cost bound on heavy vacuum world.

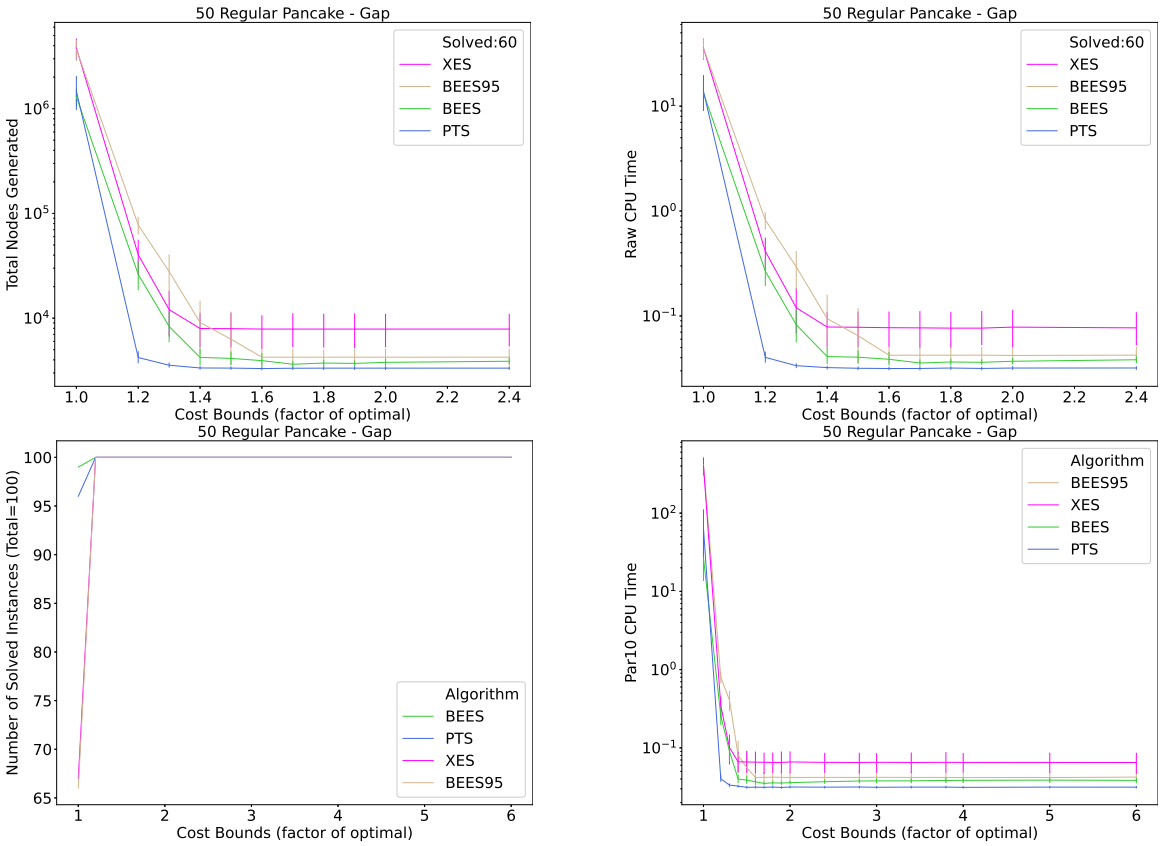


Figure 3-8: Performance measurements as a function of the cost bound on regular pancake.

Pancake. The objective of the pancake problem (Kleitman et al., 1975; Gates & Papadimitriou, 1979; Heydari & Sudborough, 1997) is to sort a sequence of pancakes through a minimal number of prefix reversals. We use the GAP heuristic (Helmert, 2010) for all the algorithms. We also consider Heavy Pancake, where the action cost is the sum of the indices of all pancakes being flipped. We use 100 randomly generated instances.

For regular pancake results shown in Figure 3-8, PTS performs well while $\widehat{\text{PTS}}$ performs poorly, with BEES, BEES95, and XES in between. This is the worst domain for XES, but its performance is not far from the best algorithm. For heavy pancake shown in Figure 3-9, XES performs similarly to $\widehat{\text{PTS}}$ and slightly worse than PTS, while BEES and BEES95 are much worse.

Racetrack. The Racetrack domain (Barto et al., 1995) is similar to the grid pathfinding problem with inertia. We use the track map that was created by Hansen and Zilberstein (Hansen & Zilberstein, 2001). Each action modifies the acceleration of the agent by -1, 0, or 1 in both the horizontal and

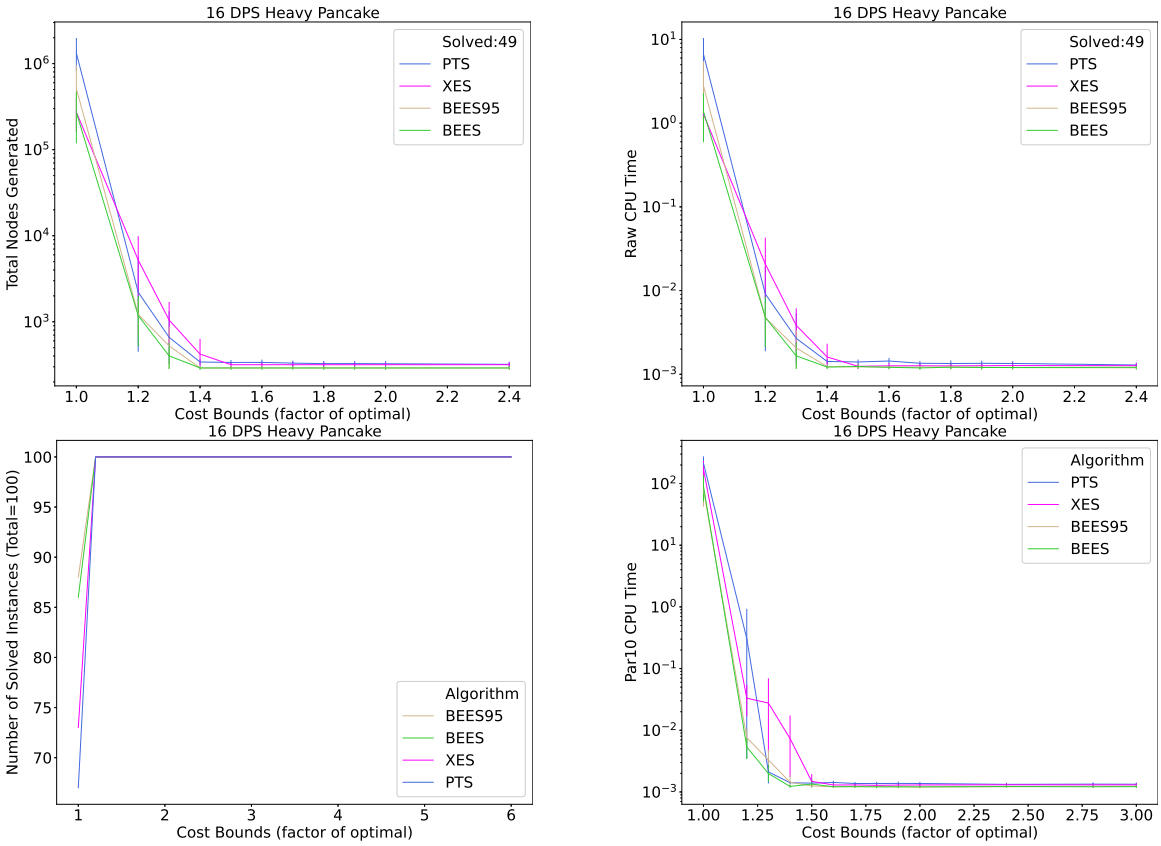


Figure 3-9: Performance measurements as a function of the cost bound on heavy pancake.

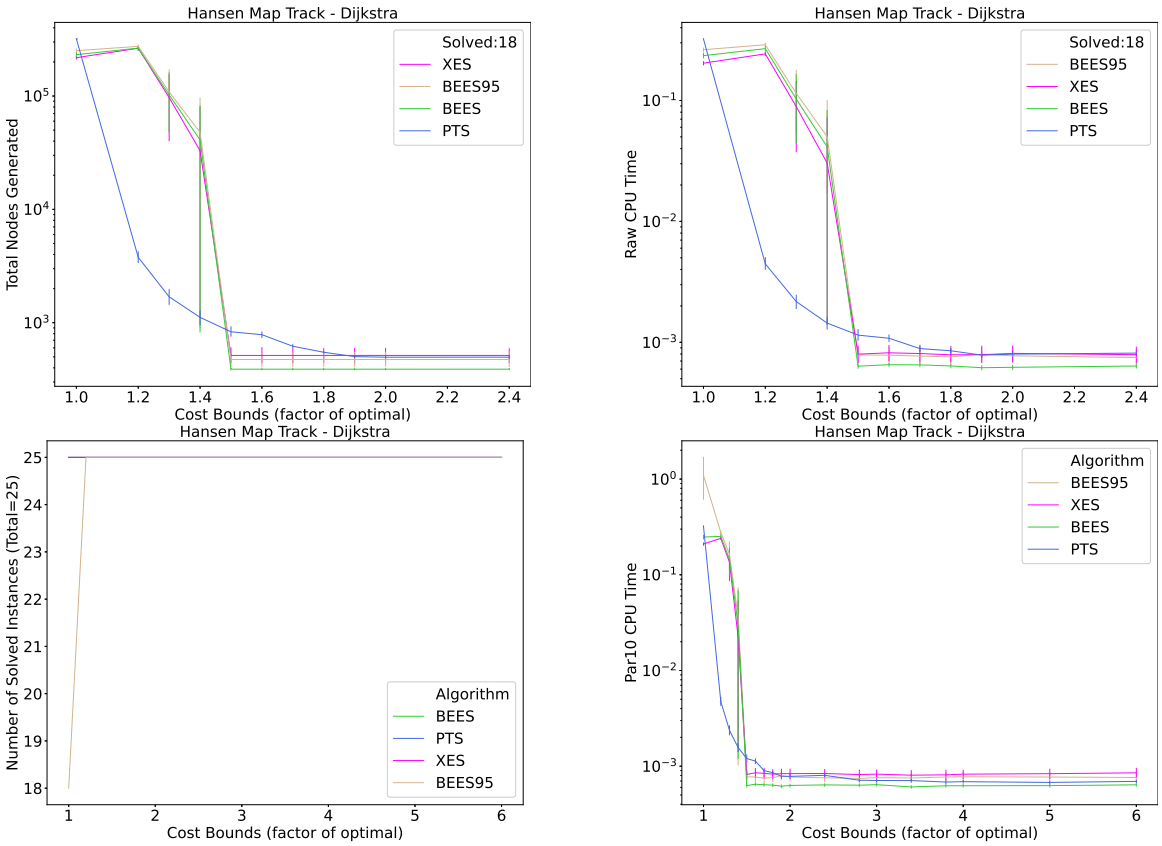


Figure 3-10: Performance measurements as a function of the cost bound on hansen map racetrack with Dijkstra heuristic.

vertical directions, making for a total of 9 distinct actions. The admissible heuristic function is the maximum, either horizontally or vertically, of the distance to the goal divided by an estimate of the maximum achievable velocity (MAV) in that dimension. MAV could possibly be attained given the domain size. We note the map width as w, and map height as h. The MAV value in the horizontal dimension, note as MAV_x, is the largest integer value such that $MAV_x(MAV_x + 1)/2 \leq w$ is still true. Similarly, MAV value in the vertical dimension, note as MAV_y, is the largest integer value such that $MAV_y(MAV_y + 1)/2 \leq h$ is still true. Both a precomputed Dijkstra distance function and a weaker Euclidean distance function are used. We created 25 instances with starting positions chosen randomly among those cells that were at least 90% of the maximum distance from a goal.

On both the Barto and Hansen maps of the Racetrack domain, most algorithms scale similarly when using the accurate Dijkstra heuristic (shown in Figure 3-10 and Figure 3-11), with PTS and $\widehat{\text{PTS}}$ being slightly better when the bound is very tight and $\widehat{\text{PTS}}$ being worse when the bound is

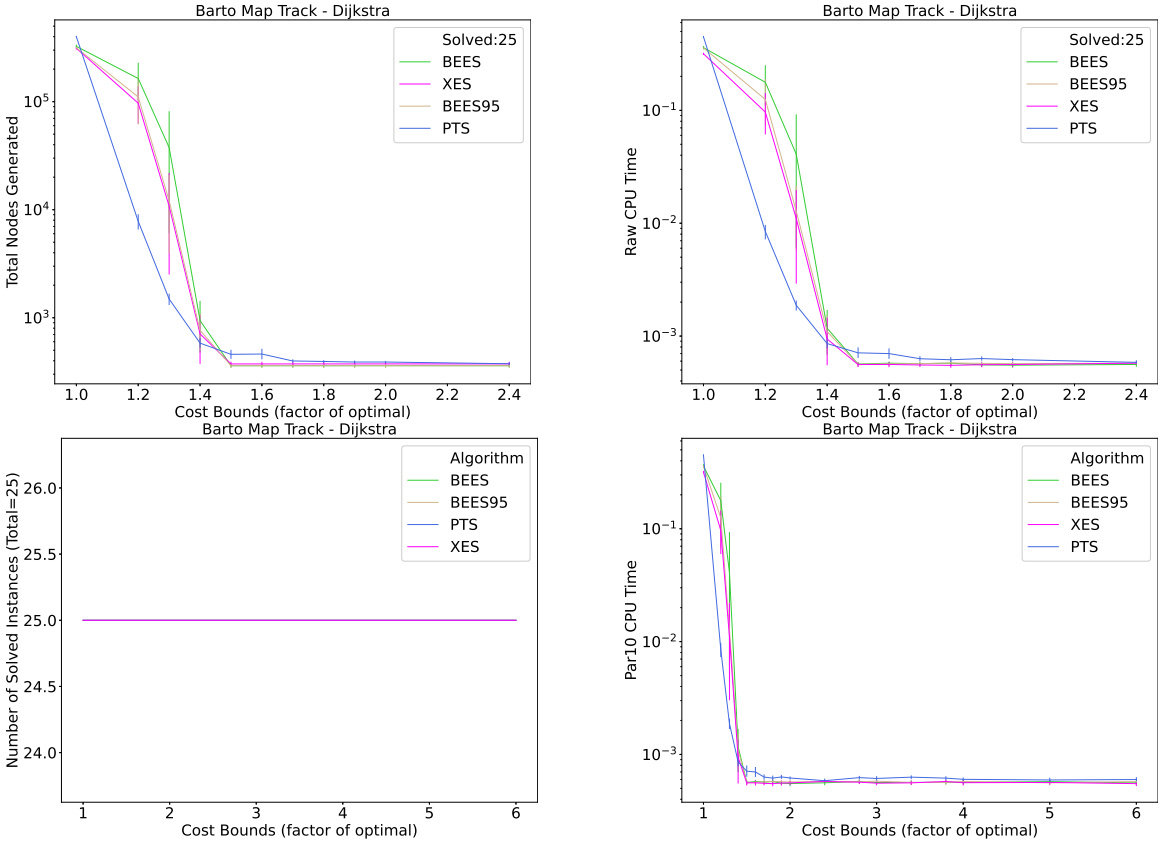


Figure 3-11: Performance measurements as a function of the cost bound on barto map racetrack with Dijkstra heuristic.

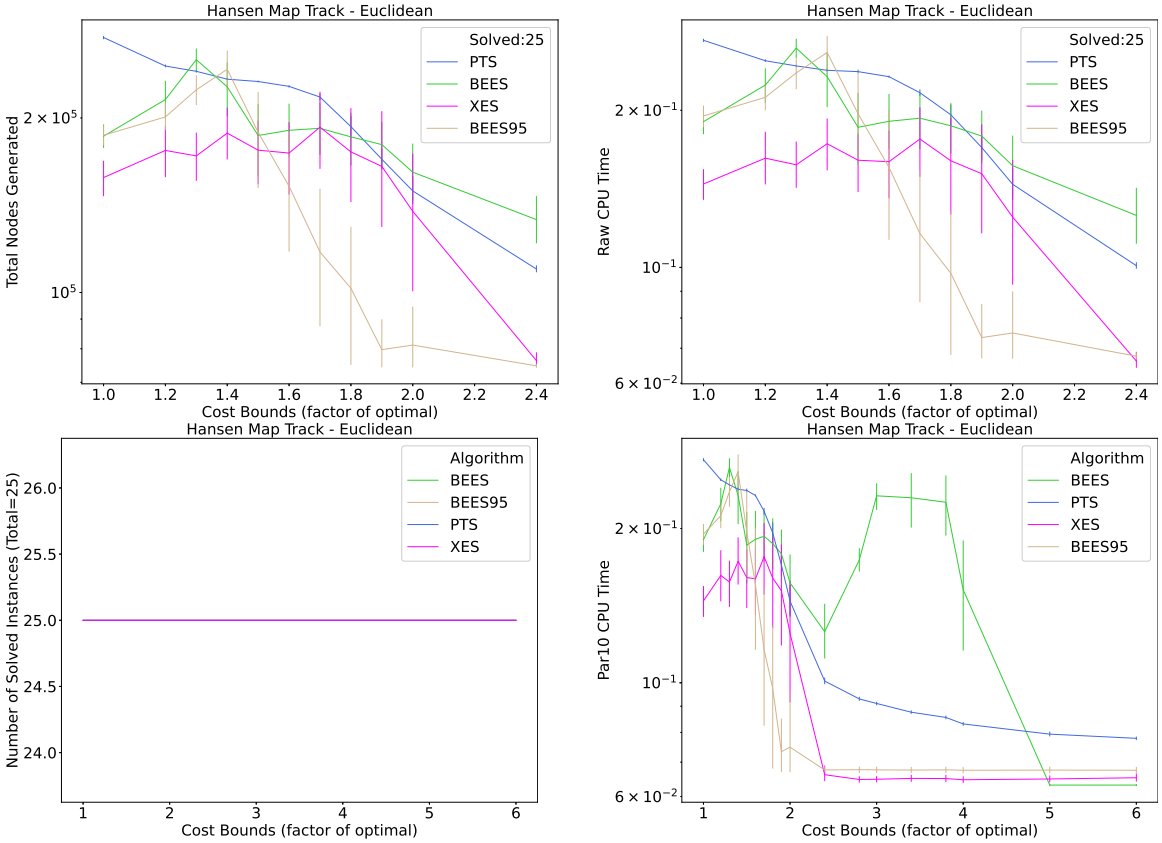


Figure 3-12: Performance measurements as a function of the cost bound on hansen map racetrack with euclidean heuristic.

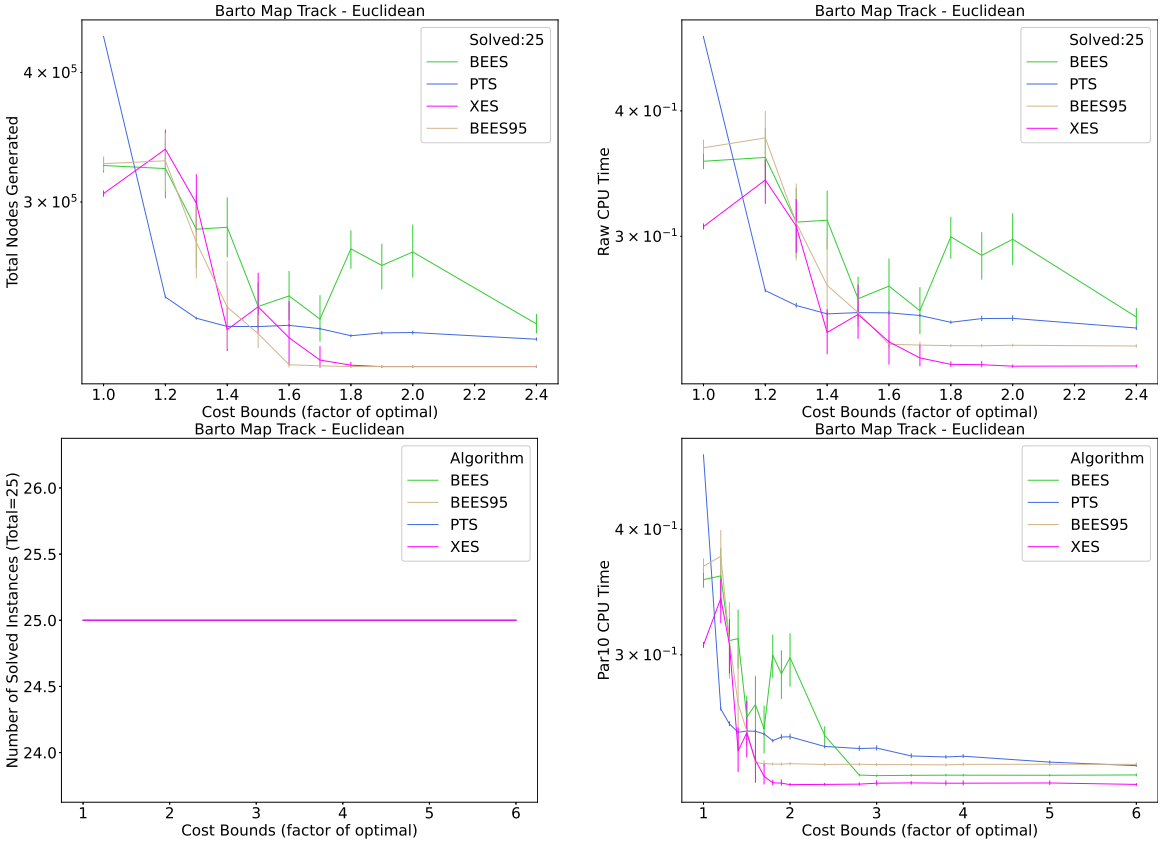


Figure 3-13: Performance measurements as a function of the cost bound on barto map racetrack with euclidean heuristic.

Coverage	GBFS	PTS	$\widetilde{\text{PTS}}$	BEES	BEEPS	BEES95	BEEPS95	XES
Agricola (20)	**1**	0	0	0	0	0	0	0
Caldera (20)	8	10	11	10	10	12	11	**13**
Caldera-split (20)	**4**	2	2	2	2	2	2	2
DataNetwork (20)	2	0	0	3	3	3	3	**4**
Nurikabe (20)	4	10	7	10	10	**11**	9	9
Settlers (20)	4	5	7	10	10	**11**	**11**	**11**
Snake (20)	4	**5**	**5**	4	**5**	4	**5**	**5**
Spider (20)	7	**11**	9	10	10	10	9	9
Termes (20)	11	9	6	11	10	11	10	**13**
Sum (180)	45	52	47	60	60	64	60	**66**
Exp. ($*10^3$)	1.93	3.93	6.75	2.10	2.62	2.25	2.24	**1.77**
Search time (s)	**1.69**	4.11	7.20	2.14	2.79	2.32	2.39	1.91

Table 3-1: Coverage on the IPC'18 bounded-cost instances, and geometric means of the expansions (multiply by 10^3) and search time on commonly solved instances (last two rows).

loose. However, using the weaker Euclidean heuristic (shown in Figure 3-12 and Figure 3-13), XES and BEES95 perform better than other algorithms.

3.5.2 Planning Domains

We also implemented the bounded-cost search algorithms in Fast Downward (Helmert, 2006). The experiments were run using the lab framework (Seipp et al., 2017) on a cluster of 2.2 GHz Intel Xeon E5-2660 CPUs. The time and memory limits were set to 30 minutes and 4 GB, respectively. We use the popular FF heuristic (Hoffmann & Nebel, 2001) as the heuristic h and use the length of the relaxed plans as the search distance estimator d. All algorithms use a dual queue for preferred operators.

IPC'18 Results

Table 3-1 compares XES, BEES95, and BEEPS95 to previous bounded-cost search algorithms. We also include GBFS (with pruning on g), which was used by most planners in the IPC.

XES performs best overall with a coverage of 66, followed by BEES95 (64) and the other algorithms of the BEES family (60). The potential-based algorithms PTS and $\widehat{\text{PTS}}$ have significantly lower coverage, and the statistics on the number of expansions show that they are not as effective in guiding the search to a goal. GBFS is not competitive with the specialized bounded-cost search algorithms overall, though it can sometimes beat them if it quickly finds a cost-effective path to the goal (e.g. in Agricola and Caldera-split). On five of the nine domains, XES has the highest coverage of the considered algorithms. Across the commonly solved instances, XES also needs the fewest expansions on average, and its search time is only beaten by GBFS (which solves significantly fewer instances, but those that it can solve are solved quickly).

Satisficing Results

Table 3-2 shows the results on the instances of the satisficing tracks using the cost bounds from Planning.Domains. Consistent with the results on the IPC'18 bounded-cost instances, XES clearly outperforms the other considered algorithms: on 14 of the 48 domains it solves strictly more instances than any other algorithm, and on further 14 domains XES has the shared best coverage. Our new BEES variants BEES95 and BEEPS95 show small but relatively consistent improvements over their respective base algorithms. While PTS and $\widehat{\text{PTS}}$ show good performance in some individual domains (e.g., Floortile and Freecell), they again lag behind the other bounded-cost search algorithms overall.

Figure 3-14 shows the normalized coverage (fraction of solved instances averaged across all domains) as the cost bounds are multiplied by increasing factors. The relative strength of the algorithms remains mostly consistent across the different bounds, with XES being the best performing algorithm for all considered values, and the specialized bounded-cost search algorithms still have a significant advantage over GBFS with pruning even as the cost bound is relaxed.

Coverage	GBFS	PTS	$\widehat{\text{PTS}}$	BEES	BEEPS	BEES95	BEEPS95	XES
Airport (49)	26	24	24	31	31	**32**	**32**	31
Assembly (30)	17	18	18	25	24	**30**	29	**30**
Barman (40)	5	0	0	6	6	9	9	**10**
Blocks (35)	19	26	26	30	26	30	26	**31**
Cavediving (8)	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**
Childsnack (6)	1	0	0	0	0	0	0	**3**
CityCar (13)	1	4	4	4	4	**5**	**5**	4
Depot (21)	6	13	13	**14**	13	12	13	13
DriverLog (20)	10	**19**	18	18	18	18	**19**	15
Elevators (39)	7	12	12	22	22	**24**	**24**	**24**
Floortile (27)	6	**20**	**20**	8	8	9	8	9
Freecell (80)	20	**65**	57	37	47	39	58	41
GED (20)	0	**1**	0	0	0	**1**	**1**	**1**
Grid (5)	1	3	3	3	3	3	3	**4**
Gripper (20)	7	8	8	7	8	7	8	**12**
Hiking (18)	8	**18**	16	15	17	14	17	16
Logistics (63)	33	46	38	51	50	52	**53**	52
Maintenance (17)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Miconic (439)	342	362	427	403	426	397	426	**437**
Movie (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**
Mprime (35)	29	**34**	33	33	33	**34**	33	33
Mystery (19)	17	**19**	**19**	**19**	**19**	**19**	**19**	**19**
Nomystery (18)	4	16	16	**18**	14	**18**	14	17
Openstacks (98)	33	31	41	53	54	56	56	**58**
Opt. Telegr. (4)	**4**	2	2	2	2	**4**	**4**	**4**
Parcprinter (40)	26	25	20	**29**	27	24	20	24
Parking (40)	4	**21**	10	11	12	16	13	15
Pathways (57)	13	20	**27**	22	22	25	22	24
Pegsol (35)	32	**35**	**35**	34	**35**	34	**35**	**35**
Philosophers (45)	**45**	11	5	11	11	11	11	14
Pipes-notank (46)	20	41	42	**43**	**43**	42	42	**43**
Pipes-tank (45)	15	29	28	28	32	26	31	**33**
PSR (116)	107	111	110	111	110	111	110	**113**
Rovers (40)	12	20	18	25	25	27	25	**30**
Satellite (36)	20	20	20	**23**	21	21	**23**	**23**
Scanalyzer (28)	20	16	17	20	18	**22**	17	21
Schedule (150)	36	51	47	62	73	60	75	**89**
Sokoban (30)	24	**29**	**29**	**29**	**29**	**29**	**29**	**29**
Storage (28)	14	19	20	20	20	16	18	**22**
Tetris (18)	**7**	4	3	4	4	2	2	3
Thoughtful (20)	6	10	10	8	10	9	9	**12**
Tidybot (17)	3	11	11	**12**	**12**	11	11	**12**
TPP (30)	10	16	13	20	19	15	18	**24**
Transport (53)	7	9	10	15	15	**19**	**19**	17
Trucks (31)	18	30	30	30	29	**31**	**31**	30
VisitAll (37)	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Woodw. (31)	18	**24**	11	19	19	21	19	22
Zenotravel (20)	8	14	13	13	13	**16**	**16**	14
Sum (2147)	1098	1344	1361	1425	1461	1438	1490	**1550**
Normalized (%)	44.8	58.5	55.9	59.9	59.8	61.6	61.9	**66.1**
Expansions	1961	1135	1787	401	491	**365**	485	369
Search time (s)	0.53	0.40	0.57	0.19	0.23	**0.18**	0.24	**0.18**

Table 3-2: Coverage on the IPC satisficing instances. The normalized overall coverage corrects for the different numbers of instances per domain. The last two rows show the geometric means of the expansions and search time on commonly solved instances.

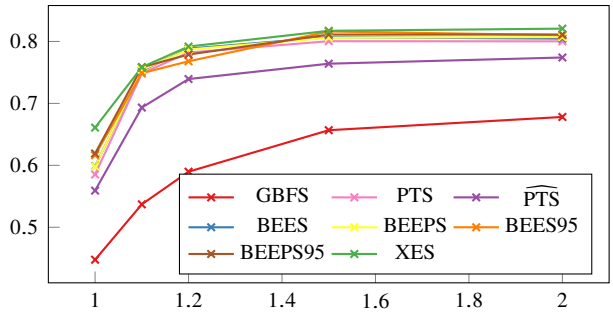


Figure 3-14: Normalized coverage as the cost bound is increased.

3.5.3 Experiment Summary

The results on benchmarks from both planning and heuristic search show that XES performs consistently well across almost all tested domains. It is often better than previous state-of-the-art bounded-cost search algorithms. The distributionally-enhanced BEES variants BEES95 and BEEPS95 improve over BEES and BEEPS in most domains. One advantage of XES over these BEES variants is that it is less reliant on the probability estimate being accurate: If the probability estimate consistently over- or underestimates, then the expected effort values are affected equally and the ordering of the open list does not change much, while the focal list is either over- or underutilized in the BEES algorithms. There are some domains where PTS and $\widehat{\text{PTS}}$ work well, but their performance is less robust to weaker heuristics.

3.6 Conclusions

By representing uncertainty, XES optimizes search effort in a simple formal model and empirically appears more robust than other state-of-the-art bounded-cost algorithms in heuristic search benchmarks. We show that even in single-agent deterministic domains, representing an algorithm's uncertainty about its estimates of properties of the unexplored portions of the search space can be a valuable tool. As in Chapter 2, this again shows us how heuristic search can benefit from reasoning about uncertainty.

CHAPTER 4

Metareasoning for Situated Planning

In online planning, planning happens concurrently with execution. Under the formulation of planning as heuristic search, when the planner commits to an action, it re-roots its search tree at the node representing the outcome of that action. For the system to remain controlled, the planner must commit to a new action (perhaps a no-op) before the previously chosen action completes. This time pressure results in a real-time search. In this time-bounded setting, it can be beneficial to commit early, in order to perform more lookahead search focused below an upcoming state. For example, let's think of a simplified autonomous driving scenario that our agent is currently driving on a boring highway. It can be beneficial for the agent to commit all the actions during the long boring drive and using the long execution time to search for a plan for the more complicated downtown situation in the future. In this chapter, we propose a principled method for making this commitment decision. Our approach again relies on representing value uncertainty. Our experimental evaluation shows that our scheme can outperform previously-proposed fixed strategies. This gives us the third example of how representing uncertainty can benefit heuristic search.

This work appeared in the proceedings of the ICAPS-21 Workshop on Integrated Planning, Acting, and Execution (IntEx-21) (Gu, Ruml, Shperberg, Shimony, & Karpas, 2021). All authors did joint work on algorithm design and theoretical analysis. Tianyi Gu implemented the algorithm in the domain-specific framework and conducted experiments in traditional search domain.

4.1 Introduction

Many applications of planning involve time pressure. Often, we want to achieve the goal as soon as possible, minimizing the so-called Goal Achievement Time (GAT) (Hernández, Baier, Uras, & Koenig, 2012; Kiesel, Burns, & Ruml, 2015). In situated temporal planning, external time

constraints, such as buses or trains that depart at scheduled times, cause plans to become infeasible if we take too long to plan (Cashmore, Coles, Cserna, Karpas, Magazzeni, & Ruml, 2018; Shperberg, Coles, Karpas, Ruml, & Shimony, 2021). One way to address time pressure is to design faster planning algorithms. But many planning problems are inherently intractable. The most direct way to address planning under time pressure is to allow actions to begin executing before a complete plan has been found. Further planning can then overlap with action execution. The fundamental question in such online planning settings is: when should the planner commit to an action?

Existing methods offer simple fixed answers to this question. Some methods use a fixed amount of lookahead for every action selection decision and commit to exactly one action given this lookahead. Others commit to multiple actions, even the entire sequence of actions leading all the way to the search frontier. In this chapter, we develop a principled approach to action commitment that uses heuristic information to assess the planner's uncertainty about action values. This uncertainty then drives the decision of whether to commit to an action or whether to perform additional lookahead before deciding. Our results indicate that the approach has promise, as it outperforms previous non-adaptive strategies in three challenging scenarios.

4.2 Background

4.2.1 Problem Setting

Our problem setting requires the system to be controlled at all times. That is, some action must be executing at any given time, even if it is just a no-op that leaves the state unchanged. (Some domains, such as fixed-wing aircraft control, do not have no-op actions.) We make the additional simplifying assumptions that actions are serial and that the world is fully observable and deterministic. Thus, we have a planner searching for a sequence of actions under the constraint that at all times at least one action (beyond those that have already completed) has been computed, committed to, and has begun execution. The objective of the system is to achieve a goal as soon as possible. Because we are addressing concurrent planning and execution, we used GAT as our main evaluation metric. This is the total time taken from the start of planning to the arrival of the agent at a goal.

We take a heuristic search perspective, in which planning explores an incrementally-generated tree of feasible action sequences, with the root of the tree representing the state resulting from the execution of all actions that have been committed to up to now. The planner is allowed to commit to actions earlier than required, in order to allow it to re-root the tree at a deeper node, thereby focusing the search later on into the future. A commitment queue records all the committed actions. How and when to make such additional commitments, so as to reduce the expected time to reach the goal, is the focus of this chapter. Following Russell and Wefald (1991), we aim to pose and solve this question as a decision-theoretic metareasoning problem. However, even this limited focus is too general to formalize, hence we make additional metareasoning assumptions about the search process:

1. The order of decision in the planner is a fixed search tree structure, from early actions to later actions.

2. No replanning is permitted after action commitment, a decision to commit to an action in the sequence means that it will eventually be executed in the order specified.

3. We may re-start search at a new state if necessary, for example, if the controlled system departs from our assumption of determinism.

4. The only question we address is when to 'reroot the tree' at a successor of the root, that is, should we do this before it is necessary?

5. We assume a given expansion strategy that is not modified by the commitment strategy, other than by pruning the parts of the search tree inconsistent with the action commitments.

Note that the metareasoning assumptions are meant to define the constraints on the decision-making at the metareasoning level, rather than representing assumptions about the domain or the planner. They are used to define the distributions and utilities. Nevertheless, in an actual implementation the planner may deliberately act in a way that does not conform to the assumptions, especially when it is obvious that better performance can be achieved by violating the assumptions. For example, it is possible, due to several commitment decisions, to get a commitment queue containing

a sequence of actions that makes the agent walk in a loop. In such cases, if this is observed before beginning to execute these actions, it may decide to remove such a loop from the action queue, even though this may not be admitted by the metareasoning assumptions.

Periodically during the search, perhaps after each expansion or periodically after a set of expansions, a metareasoning process decides between two options:

- commit to the current seemingly-best top-level action now and re-root the search tree accordingly, or
- postpone the commitment and continue the current search.

Note that if we always decide to postpone, eventually action execution will reach the current root node state and force us to commit to a next action.

4.2.2 Previous Work

The seminal work of Korf (1990) defined the problem setting of single-agent real-time search, in which a fixed number of expansions (or equivalently, amount of time) is allowed for lookahead node expansions, after which the search must commit to the next action to take and re-root the search tree. His RTA* and LRTA* algorithms back up h values from the lookahead frontier to inform the action choice, caching the backed up values at every node to allow the heuristic information to become more accurate over time and provably prevent the search from becoming stuck in infinite loops. (The LRTA* variant converges to the optimal h values.) These algorithms were designed to be simple and amenable to analysis. They commit only to a single action, which means that the lookahead of one iteration can have significant overlap with the nodes visited in the previous iteration, depending on the state space connectivity and the heuristic function.

The widely popular LSS-LRTA* algorithm (Koenig & Sun, 2008) takes a different approach, committing to the entire sequence of actions leading to the most promising frontier node. This reduces the re-generation of nodes seen during the previous lookahead and reduces the overall overhead of the search per executed action, but note that it also commits the agent to certain actions, such as those at or near the frontier, for which little lookahead has been performed and for which the heuristic value of their resulting successor state is their only attractive attribute.

The Dynamic $\hat{f}$ algorithm (Kiesel et al., 2015) modifies LSS-LRTA* in two ways. First, rather than idling the planner for $k - 1$ time steps after committing to k actions, Dynamic $\hat{f}$ uses the entire time until all the committed actions have finished executing to perform lookahead search. The amount of lookahead is thus adjusted dynamically, rather than being fixed from the start. This often results in the next iteration having a sequence of more than k actions to the best node on the search frontier, leading to a virtuous circle of larger and larger lookahead. Second, rather than expanding the frontier node with the lowest f value, the algorithm computes an inadmissible heuristic $\hat{h}$, which when added to g yields the inadmissible (but possibly more accurate) total plan cost estimate $\hat{f}$. By selecting the node with lowest $\hat{f}$, Dynamic $\hat{f}$ tries to avoid being tempted by shallow nodes whose admissible f values are low merely because they haven't been explored as deeply as others.

The stark contrast between the two fixed commitment strategies of LRTA* (one action) and LSS-LRTA* and Dynamic $\hat{f}$ (all the way to the frontier) raises the question of whether a principled adaptive strategy can be found to decide when to commit to an action. The first approach in this direction was Decision-Theoretic A* (DTA*) (Russell & Wefald, 1991), which attempts to optimize GAT by periodically deciding whether to continue the current lookahead search or commit to an action and re-root the tree. This is done by estimating whether the improvement in decision quality, measured by reduction in plan length, that is likely to result from further search would outweigh the time required to do the further search itself. In the implementation used for their experiments, training data was used to gather statistics on how often, and by how much, heuristic estimates tend to change as a results of further search. DTA* is not a real-time search algorithm, in that it does not respect or consider a time bound on lookahead. There is no requirement that the system constantly be executing an action and it is always permissible to deliberate further. Thus DTA* is capable of emulating A* and planning all the way to a goal before committing to its first action. DTA* is based on the less-performant depth-based lookahead of RTA* rather than the f-based lookahead of LSS-LRTA*, but it pioneered the deliberative metareasoning approach to action commitment.

The Mo'RTS algorithm of O'Ceallaigh and Ruml (2015) is basically a modification of DTA* into a true real-time search algorithm based on LSS-LRTA*. We focus here on its action commitment strategy, called $\hat{f}_{PMR}$. It assumes that a no-op 'identity' action is available in every state, which

allows the planner to continue searching from the same root. Once the path from the root to the most promising frontier node has been identified, $\hat{f}_{PMR}$ considers each node in turn, asking whether additional search would be worthwhile, and stopping at the first node for which this appears true. However, $\hat{f}_{PMR}$ does not offer a principled way to evaluate this decision at each node. It estimates the benefit of search as the expected reduction in time-to-goal resulting from more certain estimates of action cost, which seems reasonable. However, it is much harder to asses the costs of stopping the re-rooting process short of the frontier. The $\hat{f}_{PMR}$ method uses the time required to regenerate the path from the node to the frontier, which, as the authors note, is not particularly reasonable because this repeated work would likely happen concurrently with execution, not affecting the goal achievement time directly at all. This leaves the approach fundamentally unsatisfying.

In this work, we propose what we believe to be a more principled metareasoning scheme for action commitment, which we call Flexible Action Commitment Search (FACS). We integrate FACS into Dynamic $\hat{f}$ and assess its behavior using three challenging grid pathfinding scenarios specially designed to stress real-time search in different ways.

4.3 Metareasoning for Action Commitment

Our objective is to minimize GAT. Thus g, h, and f values in the state space will represent the duration of actions and the total utility of a final outcome is exactly the sum of action costs/durations taken to reach the achieved goal state. So optimizing f directly optimizes total utility.

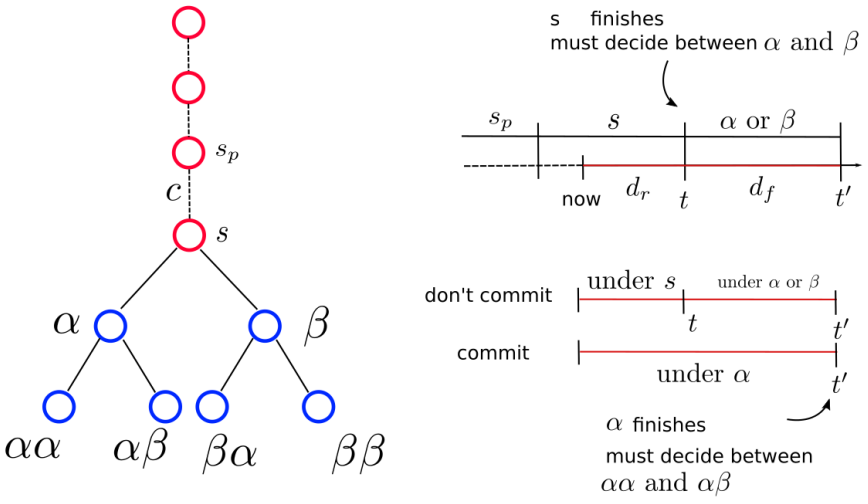
The metareasoning problem of heuristic search can be conceptualized as a POMDP in which each state represents an entire state space graph, complete with costs on every arc and h values at every vertex. To avoid confusion in this discussion, we will use the term 'vertex' for a node in the state space graph and the term 'state' for a state in the POMDP. The search does not know which exact state space graph it is dealing with, thus its situation is captured by a belief distribution over states. Every vertex expansion action results in an observation that rules out those state space graphs that are inconsistent with the vertices, action costs, and h values that are generated. The action of expanding a node is stochastic in that the search does not know in advance which new nodes, actions costs, and h values will be observed, so there are many possible belief distributions resulting from

every expansion. The action of committing to an action and re-rooting the search tree at a new vertex is deterministic, as it does not yield new information. A goal in the POMDP is a belief that has positive support only on state spaces that all share the same path from the initial vertex to a goal vertex, providing a solution to the original problem but potentially harboring remaining uncertainty about the unseen portions of the graph. A policy for the POMDP corresponds to a search strategy, as it would prescribe an action for the search to take at every reachable belief state. Solving the POMDP for a policy that, for example, minimizes expected solution length would give a heuristic search strategy that finds a solution as quickly as possible by minimizing the expected number of expansions. Approaching such a problem in practice depends crucially on exploiting structure in the h values, the arc costs, and the distance to the nearest goal.

It is not feasible to solve this POMDP, or even to find a reliable approximation of its solution using standard approximation methods. Therefore, we propose a myopic metareasoning scheme that only considers the next action commitment decision. In this formulation, one of the following two options must be chosen:

- Commit to the action with the best (least) estimated $\hat{f}$-value among all children of the current root node. We denote the node that corresponds to this action by α.
- Do not commit to α yet, and spend more time searching before deciding which action to take next.

Prematurely committing to α might reduce the quality of the solution. For example, if α leads to a dead-end and the search algorithm has failed to figured that out before committing, then it would be forced to turn around eventually, which would result in a solution with an increased cost. On the other hand, by gaining additional search time before making a consequential decision the search algorithm might be able to avoid future dead-ends or pitfalls, which would not have been possible to avoid otherwise. Thus, the utility of committing to α or not committing to α should depend in part on our certainty regarding the $\hat{f}$-value of α.



(a) (b)

Figure 4-1: Committing vs not committing.

4.3.1 The Effect of Committing

Let $P_s^d(x)$ be the predicted probability of having the belief that $\hat{f}(s) = x$ given d more node expansions of search under node s. Denote by X_s^d the random variable distributed as P_s^d.

We begin with several additional simplifying assumptions:

1. Each node has exactly two children (a branching factor of 2), α and β, where α is the node with the highest expected utility (lowest expected $\hat{f}$-value); we will relax this assumption in the next subsection.

2. The time d_f required to fully execute an action is identical for all actions; this assumption will also be relaxed later.

3. When searching under a node s, the search time is evenly divided among all of its children.

4. The X_s^d random variables are independent for all d and s.

Under the above assumptions, we can now estimate the utility of committing to and of not committing to α. In Figure 4-1(a), we show the current search tree rooted at node s. The available search time consists of the remaining time d_r induced by previous commitments and d_f, the time required to execute a full action α or β (see the the top red time line in Figure 4-1(b)). By committing

to α, the agent would be able to invest all of the available search time to search under the children of α (the bottom red time line in Figure 4-1(b), starting with the word “commit"). We denote the children of α as $\alpha\alpha$ and $\alpha\beta$ (again, see the search tree in Figure 4-1(a)). Since we assume that search time is evenly divided between $\alpha\alpha$ and $\alpha\beta$, each of them receives a search duration budget of $d = \frac{d_r + d_f}{2}$. Thus, the utility estimate of committing to α (denoted as U_{commit}) can be defined as the expectation of the minimum $\hat{f}$-value of $\alpha\alpha$ and $\alpha\beta$, after searching d time units under each of them:

$$U_{\text{commit}} = \mathbb{E}\left[\min(X^d_{\alpha\alpha}, X^d_{\alpha\beta})\right] \tag{4.1}$$

If the agent chooses not to commit yet (commit later), the remaining time d_r will be used to search under current root s (see the middle red time line in Figure 4-1(b), starting with the words “don’t commit"). Thus half of d_r ($\frac{d_r}{2}$) will be used to search under each child of the root. Even though α is initially estimated to have the lowest $\hat{f}$-value among the children of the root, this estimation can change after searching for $\frac{d_r}{2}$ time under α and β. In essence, the new $\hat{f}$-value estimation of α induced by the additional search can be greater than the new $\hat{f}$-value estimation of β. Thus, the rest of the time line (d_f) is used for searching under whichever child of the root is judged most promising at that time (again, see the middle red time line in Figure 4-1(b), starting with the words “don’t commit"). As a result, the search duration under each grandchild of the current most promising child (either α or β) will be $d' = \frac{\frac{d_r}{2} + d_f}{2}$. In our simplification, the branching factor is 2, so:

Case 1: after $\frac{d_r}{2}$ time spent searching under α and β, we will believe that $\hat{f}(\alpha) \leq \hat{f}(\beta)$. In this case, the rest of the search time would be invested under α:

$$U_\alpha = \mathbb{E}\left[\min(X^{d'}_{\alpha\alpha}, X^{d'}_{\alpha\beta})\right] \tag{4.2}$$

Case 2: after $\frac{d_r}{2}$ time spent searching under α and β, we will believe $\hat{f}(\alpha) > \hat{f}(\beta)$. Symmetrically to the previous case, here the rest of the search time would be invested under β:

$$U_\beta = \mathbb{E}\left[\min(X^{d'}_{\beta\alpha}, X^{d'}_{\beta\beta})\right] \tag{4.3}$$

Then, we can estimate the overall utility of committing later by weighting the probability of α and β to become the most-promising nodes after the initial search time with their corresponding utilities.

The probability of α becoming the most-promising child (choosing to commit to α) can be defined as follows:

$$P_{\text{choose } \alpha} = P((X_{\alpha}^{\frac{d_r}{2}} - X_{\beta}^{\frac{d_r}{2}}) < 0) \tag{4.4}$$

The utility of not committing at t' denoted $U^{t'}_{\text{don't commit}}$ can be estimated as:

$$U_{\text{don't commit}} = P_{\text{choose } \alpha} \cdot U_{\alpha} + (1 - P_{\text{choose } \alpha}) \cdot U_{\beta} \tag{4.5}$$

Using these equations, the metareasoning scheme simply needs to compute the utility of committing to α (Equation 4.1) and not committing to α (Equation 4.5), and to choose the metalevel action with the highest utility (lowest expected cost).

4.3.2 A Conceptual Example

In Figure 4-1, suppose that after the search, we obtain the expected cost under each leaf node, so we have $\hat{f}_{\alpha\alpha} = 3$, $\hat{f}_{\alpha\beta} = 5$, $\hat{f}_{\beta\alpha} = 4$, $\hat{f}_{\beta\beta} = 6$. And we also have $\hat{f}_{\alpha} = 3$, $\hat{f}_{\beta} = 4$ simply by backing-up from their best child node $\alpha\alpha$ and $\beta\alpha$ respectively. We are at the root node s and want to decide whether to commit to the current best action and re-root the search at α or not commit and keep searching under s. Suppose further that the expansion rate is 10 expansions per action duration, and that the action c leading to s is currently 5 expansions from completing execution. In this case, $d_r = 5$ and $d_f = 10$.

If we choose to commit, the total 15 expansions will be used to perform search under α, so $\alpha\alpha$ and $\alpha\beta$ both gain 7.5 expansions under our even division search time assumption. Now we can obtain the belief distribution for the future $\hat{f}$-value after search via Equation 4.10 (discussed below): $X^{7.5}_{\alpha\alpha} \sim \mathcal{N}(3, 0.4)$, $X^{7.5}_{\alpha\beta} \sim \mathcal{N}(5, 2.0)$. Then by applying Equation 4.1, we get the $U_{commit} = 3.2$. This can be calculated directly using the closed-form formula for the minimum of two normally distributed random variables (Nadarajah & Kotz, 2008).

If we choose not to commit, we have two search phases: before and after c completes. In the first phase, we still search sub-trees under both α and β, so both gain $d_r/2 = 2.5$ expansions. Because the system can not be left uncontrolled, we are forced to commit when c completes. So in the second phase, after c completes, the search will only expand nodes either under α or β with

$df = 10$ expansions. Thus we have $d' = (2.5 + 10)/2 = 6.25$ expansions for each leaf node. To compute U_α, now we can again obtain the belief distribution of future $\hat{f}$-value by Equation 4.10: $X^{6.25}_{\alpha\alpha} \sim \mathcal{N}(3, 0.2)$, $X^{6.25}_{\alpha\beta} \sim \mathcal{N}(5, 1.5)$. Equation 4.2 can give us $U_\alpha = 3.1$. The same computation can be applied to the β subtree to get $X^{6.25}_{\beta\alpha} \sim \mathcal{N}(4, 0.1)$, $X^{6.25}_{\beta\beta} \sim \mathcal{N}(6, 1.3)$, and $U_\beta = 4.2$. By Equation 4.4, say we get $P_{choose\alpha} = 0.7$, then we can have $U_{don'tcommit} = 0.7\times3.1+0.3\times4.2 = 3.43$. In this case, the meta-level decision is to commit since it results in the lowest expected cost of 3.2.

4.3.3 Relaxing the Assumptions

In order to relax the branching factor 2 and the identical action duration assumptions, we make the following modifications. Let $Children(x)$ be the set of children of node x, let $b = |Children(root)|$, and let d_a be the duration of action a. First, the search times d and d' needs to be updated with respect to b as follows: $d = \frac{d_r+d_\alpha}{b}$, $d'(a) = \frac{\frac{d_r}{b}+d_a}{b}$. Note that now d' is a function of the action chosen to be taken from the root. Then, the utility functions need to be updated. The utility of committing (Equation 4.1) should be generalized to:

$$U_{\text{commit}} = \mathbb{E}\left[\min_{c\in Children(\alpha)} X_c^d\right] \tag{4.6}$$

The utility of searching d' time under node c (generalization of equations 4.2 and 4.3):

$$U_c = \mathbb{E}\left[\min_{c'\in Childrenc} X_{c'}^{d'(c)}\right] \tag{4.7}$$

The probability of choosing node c after searching d' time under each child of the root is:

$$P_{\text{choose } c} = P(\underset{c'\in Children(root)}{\arg\min} X_{c'}^{\frac{d_r}{b}} = X_c^{\frac{d_r}{b}}) \tag{4.8}$$

Thus, the utility of not committing (generalization of Equation 4.5) becomes:

$$U_{\text{don't commit}} = \prod_{c\in Children(root)} P_{\text{choose } c} \cdot U_c^{t'} \tag{4.9}$$

4.3.4 Defining the $P_s^d(x)$ Distributions

The $P_s^d(x)$ distributions should reflect the effect of search effort under nodes given d more node expansions. Specifically, the more we search under a node, the more likely that our estimation of its

$\hat{f}$ value will change and get closer to its true value. In addition, we assume that the closer a node is to a goal, the more accurate the original estimation of its $\hat{f}$ value. Finally, the average heuristic error on the path which leads to s from the root, $\bar{\epsilon_s}$, can be used as an indicator of the quality of the $\hat{f}$ value estimation of s. Thus, the variance of $P_s^d(x)$ should grow proportionally to d and $\bar{\epsilon_s}$, and the distance-to-go estimation of s. Let $dtg(s)$ be the distance-to-go estimation of node s, ed be the average *expansion delay* which measures the number of node expansions from the moment a node is generated until it is expanded (Dionne, Thayer, & Ruml, 2011). We model $P_s^d(x)$ as a normal distribution in Equation 4.10. The mean of the distribution is the current cost-to-goal estimation, $\hat{f}(s)$. For an initial estimate of the variance we use the square of the heuristic error multiplied by the distance-to-goal of s. This uncertainty value is modeled as being reduced according to the fraction of the distance to the goal that we expect to explore using d node expansions (d divided by the expansion delay gives the distance explored). If the expected exploration depth surpasses the estimated distance-to-go, we clamp the fraction at 1.

$$P_s^d = \mathcal{N}(\hat{f}(s), (\bar{\epsilon_s} \cdot dtg(s))^2 \cdot \min(1, \frac{\frac{d}{ed}}{dtg(s)})) \tag{4.10}$$

To summarize, our Flexible Action Commitment Search (FACS) approach uses the P_s^d estimates about how the planner's beliefs about α and β will change after search in order to estimate U_{commit} and $U_{\text{don't commit}}$ and hence decide whether to commit to α or continue searching.

4.4 Empirical Evaluation

Although our approach seems to be a more principled commitment strategy, when all of its assumptions hold, than the fixed approaches in previous work, it remains to be seen how it performs in practice. In this section, we integrate FACS into Dynamic $\hat{f}$ and empirically evaluate it against three baselines: original LSS-LRTA* (i.e., commit-all), LSS-LRTA* with commit-one, and Dynamic $\hat{f}$ (i.e., commit-all with dynamic lookahead).

Figure 4-2: Schematics of grid benchmarks with tar pits.

4.4.1 Synthetic Grid Pathfinding Domain

We implemented all real-time search agents in a synthetic grid pathfinding domain using the Euclidean distance heuristic. Figure 4-2 shows a schematic view of a tricky instance of our novel variant of the classic grid pathfinding problem, specially contrived to challenge real-time search. The black areas are the obstacles. The red patches are 'tar pits', cells for which the cost of moving to an adjacent empty cell is very high (i.e., high cost for stepping out from a 'tar pit'). With this setting, there will be a high cost if an agent commits to an action that steps into a pit, as the agent would have to step out of it in order to reach the goal. Note that the admissible heuristic function does not take these costs into account. Therefore, the agent has to be very careful about its commitment decisions. The small red tar pits are very common in the left part of the map, so a search's lookahead frontier will have a high probability of including at least one, possibly even as the best node. Thus, we expect an agent with a strategy that commits all the way to the frontier to be fairly likely to step into a tar pit at some point. In the middle, we have a long empty area. Since in this area there are no traps or mazes, algorithms can safely commit and re-root the search to the frontier nodes in order to gain search time for the future. In contrast, algorithms that conservatively commit only to one action at a time and re-root the search tree at every step cannot benefit from such gains in future search time. On the right side of the map, we have a corridor setting. Again, the red region is a large tar pit, where there is a small cost of stepping into this area, but a large cost of getting out of it. If agents do not have sufficient lookahead to observe that the red region is a high cost local

minimum, they are likely to be tempted to get into this large tar pit, as it will seem to be a shorter path to the goal because we sample the goal position from the lower rows and sample the entrance of the corridor from upper rows so that the agent must go against the Euclidean heuristic to enter the upper corridor. However, with a sufficiently large lookahead, agents can detect that this tar pit is a dead-end and avoid stepping into it altogether. Therefore, we expect agents that accumulate search time (i.e., lookahead) during the middle empty region to be able to utilize it to avoid the large tar pit. In the next section, we show results for maps that only have the left-side scattered tar pits, maps that only have the right-side corridor and pit, and maps with both left-side and right-side pits.

4.4.2 Experiments

All algorithms were implemented in C++ and run on 64-bit Linux systems with 3.16 GHz Intel E8500 processors and 8 GB of RAM. We used grid maps of 50 rows and 200 columns. For each map, we set the start in the left-most column and goal in the right-most column, randomizing the row numbers of the start position and goal position to generate 100 problem instances. We used lookahead limits of 4, 10, 30, 100, and 300 expanded nodes per action (i.e., the relative search vs action execution speed), shown in the x axis of each plot in Figures 4-3-4-5. We set the cost of stepping out of a tar pit as 1,000 expansions. The y axis shows GAT, normalized as a factor of the GAT of a clairvoyant agent that immediately commits to an optimal plan without searching. Error bars show 95% confidence intervals on the mean over all the instances. The legends are sorted by the geometric mean across all lookahead limits.

Figure 4-3 shows the result of grid pathfinding problems with tar pits near the start. FACS preforms consistently close to clairvoyant across all search speeds. The commit-one strategy is also very competitive at low search speed, due to its conservative commitment strategy that helps the agent avoid stepping into tar pits. The Dynamic $\hat{f}$ and commit-all strategies are both far from optimal in this map, with dynamic $\hat{f}$ stepping less frequently into tar pits as it accumulates a slightly longer lookahead by the time it reaches the tar pit field.

Figure 4-4, shows results for maps with a corridor field near the goal. Both FACS and dynamic $\hat{f}$ perform close to clairvoyant since they take advantage of accumulating search time in the empty

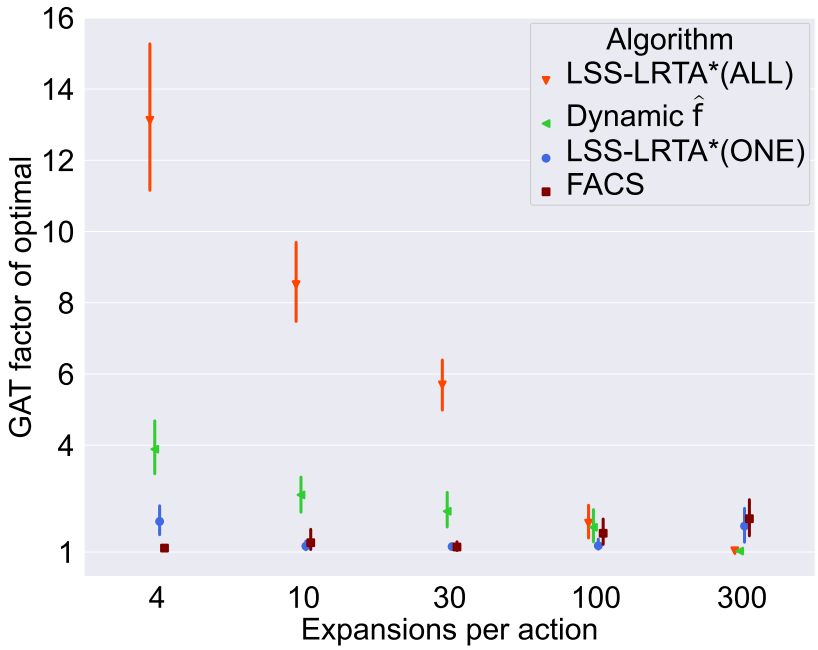


Figure 4-3: Goal achievement time as a function of search speed in grid pathfinding with tar pits near the start.

area and thus have a lookahead that is large enough to detect the dead-end and avoid stepping into the high-cost trap area. Both variants of LSS-LRTA* are unable to detect the dead-end with lookahead limit below 30.

In Figure 4-5, we show the results for maps with tar pits both near the start and near the goal. FACS is able to survive both the tar pit field and corridor field, with a conservative commitment strategy initially, adapting to an aggressive commitment strategy in the empty area, and having a sufficiently large lookahead to avoid the large tar pit when reaching the corridor.

4.5 Conclusion

In this chapter, We have suggested an approximate metareasoning scheme for action commitment geared at focusing a real-time search, in order to get additional time to search farther ahead in the search tree. Our algorithm, FACS, a basic metareasoning scheme for action commitment geared at focusing a real-time search in order to get additional time to search farther ahead in the search tree. Favorable empirical results in challenging grid pathfinding scenarios show that this approach

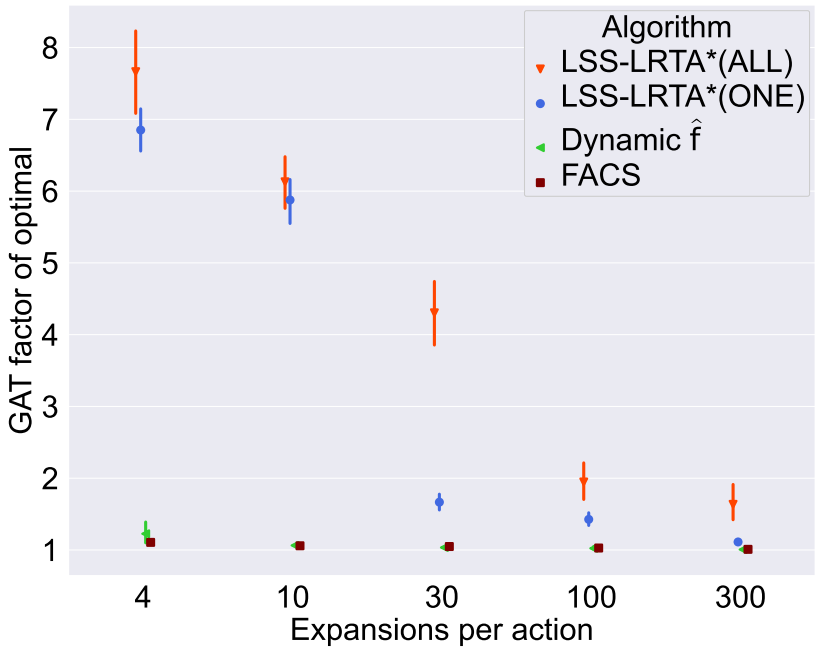


Figure 4-4: GAT with corridor and tar pit near the goal.

has promise and could lead to a principled treatment of one of the most fundamental issues on online planning and execution. This completes our last example on how representing uncertainty can benefit heuristic search.

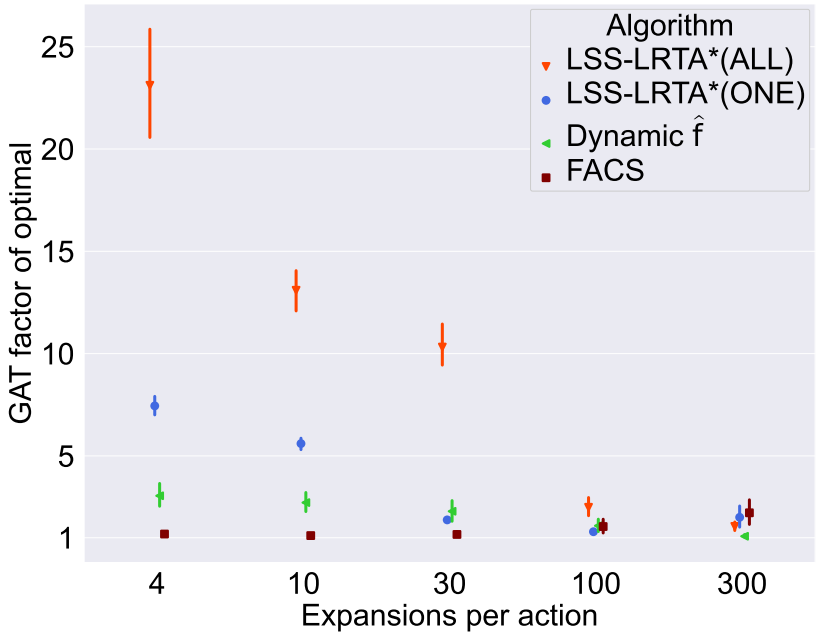


Figure 4-5: GAT with tar pits at both ends.

CHAPTER 5

Conclusion

Metareasoning can be very powerful when an agent has to deal with complex problems, especially when resources are tight. This dissertation has shown that, by reasoning about uncertainty, metareasoning can be a viable component in search and planning algorithms. One can adapt the distributional perspective from Bayesian RL to various deterministic single-agent planning settings and, in turn, deterministic planning can provide useful testbeds for methods that metareason about uncertainty during planning.

The first contribution of the dissertation provides a metareasoning-based approach for rational real-time search. Instead of relying only on single heuristic value guidance, our framework, DDNancy, can learn a data-driven cost-to-go distribution to provide more rational guidance. We also adapt other distributional RL methods (Interval Estimation and Monte-Carlo Tree Search) to real-time search. The results suggest that it can often achieve better performance than traditional approaches by considering value uncertainty.

The second contribution of the dissertation introduces a distributional-enhanced approach for bounded-cost search. Our approach, XES, uses not just point estimates but belief distributions to estimate the probability that a node will lead to a plan within the bound. Our methods can often outperform current state-of-the-art bounded-cost search methods with the help of deliberate thinking on the value uncertainty.

The third contribution of the dissertation is the design of a metareasoning method to deal with the complex online problem setting of concurrent planning and execution. Our approach, FACS, applied a rational metric for the decision-making problem of when the agent should choose to commit action and think more about the future.

We claim that heuristic search can benefit from representing uncertainty. We support our claim

by showing examples from real-time search, bounded-cost search, and situated planning. This work hopes to encourage further efforts in widening the applicability of metareasoning using uncertainty for suboptimal search and planning under time pressure.

As we have seen, there are many challenges of applying metareasoning approach. For example, how to gather useful information to represent the uncertainty to prevent the metareasoning agent reasons from misleading belief distribution? Our Nancy approach in chapter 2 shows an example of gather these belief distribution by either online or offline methods. Another challenge question is how to properly approximate the effect of planning so that the agent can decide whether worth to plan more? We give an example in chapter 4 by providing a myopic metareasoning scheme.

Metareasoning is certainly not suitable for all search problems. A metareasoning enhancement often incurs computational overhead. Whether worth it to introduce such a overhead into the algorithm really depend on the problem. As we can tell from the empirical experiments, metareasoning best applied to problems with tight resources, to problems has poor heuristic function, and to problems that are very challenging to solve.

Despite of the limitations, there are still many kinds of suboptimal search and planning methods might as well benefit from adding such a metareasoning enhancement. For example, in contract search, one can represent the uncertain of the search effort estimate to better allocate its search time given the contract deadline. Another example is anytime search, where one can benefit from representing the uncertainty of the up-coming result quality to get better anytime behavior. A third example could be bounded-suboptimal search where one have to represent uncertainty for both the solution cost and absolute cost bound. We leave these for future work.

Bibliography

Allouche, D., Barbe, S., de Givry, S., Katsirelos, G., Lebbah, Y., Loudni, S., Ouali, A., Schiex, T., Simoncini, D., & Zytnicki, M. (2019). Cost function networks to solve large computational protein design problems. In *Operations Research and Simulation in healthcare*. Springer.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time dynamic programming. *Artificial intelligence*, *72*(1-2), 81–138.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML-17)*.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*, Vol. 3 of *Optimization and neural computation series*. Athena Scientific.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), 1–43.

Bulitko, V., & Sampley, A. (2016). Weighted lateral learning in real-time heuristic search. In *Ninth Annual Symposium on Combinatorial Search*.

Burns, E., Kiesel, S., & Ruml, W. (2013). Experimental real-time heuristic search results in a video game. In *Proceedings of the Sixth International Symposium on Combinatorial Search (SoCS-13)*.

Cashmore, M., Coles, A., Cserna, B., Karpas, E., Magazzeni, D., & Ruml, W. (2018). Temporal planning while the clock ticks. In *Proceedings of ICAPS*.

Cserna, B. (2019). *Real-time Planning For Robots*. Doctoral Dissertation, University of New Hampshire.

Cserna, B., Doyle, W. J., Ramsdell, J. S., & Ruml, W. (2018). Avoiding dead ends in real-time heuristic search. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Culberson, J. C., & Schaeffer, J. (1996). Searching with pattern databases. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pp. 402–416. Springer.

Dearden, R., Friedman, N., & Russell, S. (1998). Bayesian q-learning. In *Aaai/iaai*, pp. 761–768.

Dionne, A. J., Thayer, J. T., & Ruml, W. (2011). Deadline-aware search using on-line measures of behavior. In *Fourth Annual Symposium on Combinatorial Search*.

Dobson, S., & Haslum, P. (2017). Cost-length tradeoff heuristics for bounded-cost search. In *Proceedings of the ICAPS-17 HSDIP Workshop*.

Eifler, R., Fickert, M., Hoffmann, J., & Ruml, W. (2019). Refining abstraction heuristics during real-time planning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 7578–7585.

Feldman, Z., & Domshlak, C. (2014). Simple regret optimization in online planning for markov decision processes. *Journal of Artificial Intelligence Research*, *51*, 165–205.

Ferguson, D., Howard, T. M., & Likhachev, M. (2008). Motion planning in urban environments. *Journal of Field Robotics*, *25*(11-12), 939–960.

Fickert, M., Gu, T., & Ruml, W. (2021). Bounded-cost search using estimates of uncertainty. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 1675–1681.

Fickert, M., Gu, T., Staut, L., Lekyang, S., Ruml, W., Hoffmann, J., & Petrik, M. (2020a). Real-time planning as data-driven decision-making. In *the ICAPS-20 Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL-20)*.

Fickert, M., Gu, T., Staut, L., Ruml, W., Hoffmann, J., & Patrik, M. (2020b). Beliefs we can believe in: Replacing assumptions with data in real-time search. In *Submitted to Thirty-fourth AAAI Conference on Artificial Intelligence (AAAI-20)*.

Fickert, M., Gu, T., Staut, L., Ruml, W., Hoffmann, J., & Petrik, M. (2020c). Beliefs we can believe in: Replacing assumptions with data in real-time search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, pp. 9827–9834.

Frazier, P. I., Powell, W. B., & Dayanik, S. (2008). A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, *47*(5), 2410–2439.

Gates, W. H., & Papadimitriou, C. H. (1979). Bounds for sorting by prefix reversal. *Discrete mathematics*, *27*(1), 47–57.

Good, I. J. (1971). Twenty-seven principles of rationality. In *Godambe, V.P. & Sprott, D.A. (Eds), Foundations of Statistical Inference*, pp. 108–141, Toronto: Holt, Rinehart, Winston.

Gu, T., Ruml, W., Shperberg, S., Shimony, E. S., & Karpas, E. (2021). When to commit to an action in online planning. In *Proceedings of the ICAPS-21 Workshop on Integrated Planning, Acting, and Execution (IntEx-21)*.

Hald, A. (1952). *Statistical Theory with Engineering Applications*. Probability and Statistics Series. Wiley.

Hansen, E. A., & Zilberstein, S. (2001). LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, *129*(1-2), 35–62.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, *4*(2), 100–107.

Haslum, P. (2013). Heuristics for bounded-cost search. In *Proceedings of ICAPS*.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M. (2010). Landmark heuristics for the pancake problem. In *Third Annual Symposium on Combinatorial Search*.

Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway?. In Gerevini, A., Howe, A. E., Cesta, A., & Refanidis, I. (Eds.), *Proceedings*

of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009. AAAI.

Hernández, C., Baier, J. A., Uras, T., & Koenig, S. (2012). Time-bounded adaptive a.. In *Proceedings of AAMAS*, pp. 997–1006.

Heydari, M. H., & Sudborough, I. H. (1997). On the diameter of the pancake network. *Journal of Algorithms*, *25*(1), 67–94.

Hoffmann, J., & Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, *14*, 253–302.

Horvitz, E. J. (1990). *Rational metareasoning and compilation for optimizing decisions under bounded resources*. Knowledge Systems Laboratory, Medical Computer Science, Stanford University.

Kaelbling, L. P. (1993). *Learning in Embedded Systems*. MIT Press.

Karpas, E., & Magazzeni, D. (2020). Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, *3*, 417–439.

Keller, T., & Eyerich, P. (2012). Prost: Probabilistic planning based on uct. In *Twenty-Second International Conference on Automated Planning and Scheduling*.

Keller, T., & Helmert, M. (2013). Trial-based heuristic tree search for finite horizon MDPs. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*.

Kiesel, S., Burns, E., & Ruml, W. (2015). Achieving goals quickly using real-time search: experimental results in video games. *Journal of Artificial Intelligence Research*, *54*, 123–158.

Kim, K. H., & Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of operational research*, *156*(3), 752–768.

Kleitman, D., Kramer, E., Conway, J., Bell, S., & Dweighter, H. (1975). Elementary problems: E2564-e2569. *The American Mathematical Monthly*, *82*(10), 1009–1010.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, pp. 282–293.

Koenig, S., & Sun, X. (2008). Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems*, *18*(3), 313—341.

Korf, R. E. (1985). Iterative-deepening-A*: An optimal admissible tree search. In *Proceedings of IJCAI-85*, pp. 1034–1036.

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

Li, J., Tinka, A., Kiesel, S., Durham, J. W., Kumar, T. K. S., & Koenig, S. (2021). Lifelong multi-agent path finding in large-scale warehouses. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 11272–11281.

Lieck, R., & Toussaint, M. (2017). Active tree search. In *ICAPS Workshop on Planning, Search, and Optimization*.

McMahan, H. B., Likhachev, M., & Gordon, G. J. (2005). Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of ICML*, pp. 569–576.

Mitchell, A., Ruml, W., Spaniol, F., Hoffmann, J., & Petrik, M. (2019). Real-time planning as decision-making under uncertainty. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.

Mudgal, A., Tovey, C., Greenberg, S., & Koenig, S. (2005). Bounds on the travel cost of a mars rover prototype search heuristic. *SIAM Journal on Discrete Mathematics*, *19*(2), 431–447.

Mutchler, D. (1986). Optimal allocation of very limited search resources. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence (AAAI-86)*.

Nadarajah, S., & Kotz, S. (2008). Exact distribution of the max/min of two gaussian random variables. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *16*(2), 210–212.

O'Ceallaigh, D., & Ruml, W. (2015). Metareasoning in real-time heuristic search. In *Proceedings of the Eighth Symposium on Combinatorial Search (SoCS-15)*.

Pemberton, J. C. (1995). k-best: A new method for real-time decision making. In *Proceedings of the 1995 International Joint Conference on AI (IJCAI-95)*.

Pemberton, J. C., & Korf, R. E. (1994). Incremental search algorithms for real-time decision making. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS-94)*.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial intelligence*, *1*(3-4), 193–204.

Russell, S. J., & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach* (First edition). Prentice Hall.

Russell, S. J., & Wefald, E. (1991). *Do the right thing: studies in limited rationality*. MIT press.

Schulte, T., & Keller, T. (2014). Balancing exploration and exploitation in classical planning. In *Seventh Annual Symposium on Combinatorial Search*.

Seipp, J., Pommerening, F., Sievers, S., & Helmert, M. (2017). Downward Lab. url-https://doi.org/10.5281/zenodo.790461.

Shperberg, S. S., Coles, A., Karpas, E., Ruml, W., & Shimony, S. E. (2021). Situated temporal planning using deadline-aware metareasoning. In *Proceedings of ICAPS*, Vol. 31, pp. 340–348.

Shperberg, S. S., Shimony, S. E., & Felner, A. (2017). Monte-carlo tree search using batch value of perfect information.. In *Uncertainty in Artificial Intelligence*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Silver, D., & Veness, J. (2010). Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pp. 2164–2172.

Stern, R. T., Puzis, R., & Felner, A. (2011). Potential search: A bounded-cost search algorithm. In *Twenty-First International Conference on Automated Planning and Scheduling*. Citeseer.

Strehl, A. L., & Littman, M. L. (2004). An empirical evaluation of interval estimation for markov decision processes. In *IEEE ICTAI-04*.

Thayer, J. T., Dionne, A., & Ruml, W. (2011). Learning inadmissible heuristics during search. In *Proceedings of the Twenty-first International Conference on Automated Planning and Scheduling (ICAPS-11)*.

Thayer, J. T., & Ruml, W. (2011). Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI*, Vol. 2011, pp. 674–679. Citeseer.

Thayer, J. T., Stern, R., Felner, A., & Ruml, W. (2012). Faster bounded-cost search using inadmissible estimates. In *Twenty-Second International Conference on Automated Planning and Scheduling*. Citeseer.

Thiébaux, S., Coffrin, C., Hijazi, H., Slaney, J. K., et al. (2013). Planning with mip for supply restoration in power distribution systems. In *23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*. AAAI Press.

Tolpin, D., & Shimony, S. E. (2012). Mcts based on simple regret. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12)*.

Zilberstein, S. (2008). Metareasoning and bounded rationality. In *AAAI Workshop on Metareasoning: Thinking about Thinking*, Chicago, Illinois.