Real-time Planning as Data-driven Decision-making

Maximilian Fickert[*1], Tianyi Gu[*2], Leonhard Staut[*1], Sai Lekyang[2], Wheeler Ruml[2], Jörg Hoffmann[1], Marek Petrik[2]

[1]Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2]Department of Computer Science, University of New Hampshire, USA
{fickert,hoffmann}@cs.uni-saarland.de, {gu, ruml, mpetrik}@cs.unh.edu,
s9ldstau@stud.uni-saarland.de, sai.lekyang@gmail.com

Abstract

If reinforcement learning (RL) is the use of incrementally gathered data to drive decision-making, then any heuristic search strategy is fundamentally an RL process. This is perhaps clearest in real-time planning, where an agent must select the next action to take within a fixed time bound. Even in deterministic domains, real-time 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 paper, we present new results in a line of research that explores how an agent can benefit from metareasoning about this uncertainty. Taking inspiration from prior work in distributional methods from RL, the Nancy search framework represents its uncertainty explicitly as beliefs over cost-to-go. Nancy then expands nodes so as to minimize the expected regret in case a non-optimal action is chosen. We present detailed results showing how beliefs can be informed by prior experience and we experimentally compare Nancy against both conventional real-time search algorithms like LSS-LRTA* and approaches from RL that exploit uncertainty, such as Monte Carlo tree search and Kaelbling's interval estimation. We find that Nancy generally outperforms previous methods, particularly on more difficult problems. This work illustrates how the distributional perspective from Bayesian RL can be adapted to deterministic planning settings, and how deterministic planning can provide useful testbeds for methods that metareason about uncertainty during planning.

Introduction

Some AI applications 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. Given a forward model of the domain dynamics, the planning agent incrementally plans toward a goal, trying to minimize the total cost of the resulting trajectory. (Unlike some reinforcement learning settings, we assume here that the state transition function can be applied on arbitrary states.) 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 and Sun 2008), the lookahead in step 1 is performed using A* (Hart, Nilsson, and Raphael 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. Similar methods are used in reinforcement learning (RL), such as RTDP (Barto, Bradtke, and Singh 1995). 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 paper, we advance a line of research inspired by work in RL, viewing real-time planning as a form of decision-making under uncertainty.

For example, 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. For example, consider the situation depicted in Figure 1. The figure shows the agent's current beliefs about the expected total plan cost that would be incurred by committing to the ap-

[*]These authors contributed equally to this work.

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

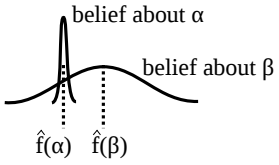

Figure 1: Should an agent expand nodes under α or β?

plicable 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 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 insights such as this derived from a reasoning under uncertainty perspective seem powerful, they are not widely applied in deterministic planning. Recently, a real-time heuristic search framework called Nancy has been proposed, that uses explicit reasoning about uncertainty to guide its search (Mitchell et al. 2019; Fickert et al. 2020). Nancy treats the traditional heuristic estimate of cost-to-go from a given state as uncertain evidence, inducing beliefs – probability distributions – over the actual remaining cost. Nancy uses a node-expansion strategy that minimizes the expected regret in case a non-optimal action is chosen.

Two alternative methods have been proposed to define the beliefs at frontier nodes. The first one is assumption-based (Mitchell et al. 2019), assuming Gaussian distributions as in prior work (O'Ceallaigh and Ruml 2015) and tuning them on-line. As these assumptions are not necessarily justified, the second method relies on data from previous search experience (Fickert et al. 2020), approximating the beliefs from data gathered in the same problem family.

In this paper, we provide further experimental results for Nancy. First, we examine the belief distributions that are learned from training data, showing in detail for the first time how heuristic error varies across planning problems. Second, we provide a comparison to methods previously proposed in RL for exploiting uncertainty in search. Monte-Carlo tree search (MCTS) methods (Kocsis and Szepesvári 2006; Keller and Helmert 2013; Feldman and Domshlak 2014; Silver et al. 2018), which originated 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 Nancy typically outperforms previous methods despite its metareasoning overhead, suggesting that it makes better use of limited node expansions. Nancy's success illustrates the strength of adopting the reasoning under uncertainty perspective from RL for resource-bounded decision-making, even in completely deterministic problem domains. It also provides a clearly defined setting might interest RL researchers that isolates the uncertainty due to computational resource bounds from that present in MDPs due to stochastic actions effects.

Previous 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.

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.

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 and Wefald 1991) and Mo'RTS (O'Ceallaigh and 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 $\hat{f}$.

MCTS algorithms such as UCT (Kocsis and 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 below, they take an approach motivated by the value of information. Lieck and Toussaint (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.

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 (Dearden, Friedman, and Russell 1998; McMahan, Likhachev, and Gordon 2005; Sanner et al. 2009; Bellemare, Dabney, and Munos 2017).

The Nancy Framework

A real-time heuristic search algorithm is made up of several parts that work together to choose the action that is most reasonable to execute next. The lookahead component determines in which direction to search, i.e. which node to expand next, in order to make the best use of the limited time. Successive lookahead steps create the local search space. The backup component runs after each lookahead step and updates the goal distance estimates of each search node based on its successors, by propagating the information from the leafs of the search tree up towards the root. In this section we give a complete description of the Nancy real-time search algorithm. Its lookahead is based on a the expected regret of making a non-optimal action choice, that is, α, the action with the smallest expected cost, turned out to have a higher actual cost than one of the alternatives β. This follows the classical characterization of risk as the sum of expected losses in a bad event (α had a higher cost than β) weighted by the probability of that event. Nancy's eponymous backup component propagates the belief distributions of the best-looking child to its parents.

Algorithm 1: Nancy

```
1  s := s_start
2  π_curr := ⟨⟩
3  while s⟦π_curr⟧ is not a goal state do
4      t := risk_lookahead(s)
5      π_curr := update_path(s, t, π_curr, π)
6      apply_next(s, π_curr)
7      backup(lss)
8  while s is not a goal state do
9      apply_next(s, π_curr)

10 fn apply_next(s, π_curr)
11     let a_0, ..., a_n be the action sequence of π_curr
12     s := s⟦a_0⟧
13     π_curr := ⟨a_1, ..., a_n⟩

14 fn update_path(s, t, π_curr, π)
15     if (π_curr = ⟨⟩ or
16         t is a goal state or
17         f̂(t) < f̂(s⟦π_curr⟧) or
18         f̂(t) = f̂(s⟦π_curr⟧) and ĥ(t) < ĥ(s⟦π_curr⟧))
       then
19         return π
20     return π_curr
```

The Components of 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. Fickert et al. (2020) prove that Nancy is complete and will always reach a goal under certain mild conditions.

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 (TLA) has an associated open list (denoted by TLA.open in the pseudo-code) that is ordered by $\hat{f}$. Before each expansion, Nancy has to pick the open list where the expansion 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

Algorithm 2: Risk-Based Lookahead

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

1 Generate *TLAs*
2 **while** *lookahead limit is not reached* **do**
3 **for** *tla in TLAs* **do**
4 Swap in $\mathcal{B}_{post}(s \mathbb{J} tla \mathbb{K})$ for $\mathcal{B}(s \mathbb{J} tla \mathbb{K})$
5 $risk[s \mathbb{J} tla \mathbb{K}] := \texttt{risk}(TLAs)$
6 Restore original $\mathcal{B}(s \mathbb{J} tla \mathbb{K})$
7 $chosen := \arg\min_{tla2TLAs} (risk[tla])$
8 $t := chosen.open.pop\_min()$
9 **if** *t is goal* **then**
10 **return** t
11 **for** $a \in A(t)$ **do**
12 Estimate and cache $\mathcal{B}(t \mathbb{J} a \mathbb{K})$ and $\mathcal{B}_{post}(t \mathbb{J} a \mathbb{K})$
13 $\texttt{push}(chosen.open, t \mathbb{J} a \mathbb{K}, \hat{f}(t \mathbb{J} a \mathbb{K}))$
14 $u := chosen.open.min()$
15 $\mathcal{B}(s \mathbb{J} chosen \mathbb{K}) := \mathcal{B}(u) + g(u)$
16 $\mathcal{B}_{post}(s \mathbb{J} chosen \mathbb{K}) := \mathcal{B}_{post}(u) + g(u)$
17 $best := \arg\min_{tla2TLAs} (\hat{f}(s \mathbb{J} tla \mathbb{K}))$
18 **return** $best.open.min()$

to arrive at a belief with minimal risk is then selected and the actual expansion follows, (Alg. 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.

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*).

Assumption-Based Nancy

Nancy's risk-based lookahead strategy relies on belief distributions over the remaining cost to a goal. To model such distributions, following O'Ceallaigh and Ruml (2015), we first build Gaussian distributions centered on $\hat{f}$ with a variance proportional to the difference between a node's $\hat{f}$ and f values:

$$B(n) \sim \mathcal{N}\Big(\hat{f}(n), \Big(\frac{\hat{f}(n) - f(n)}{2}\Big)^2\Big)$$

The mean value $\hat{f}(n)$ is estimated using one-step heuristic error estimate (Thayer, Dionne, and Ruml 2011). The variance model reflects the common assumption that heuristics are more accurate as one approaches a goal, because the difference between $f(n)$ and $\hat{f}(n)$ is proportional to $d(n)$, the estimated remaining search distance (number of steps-to-go). In this way, search experience is used to continually adjust the algorithm's skepticism of its heuristic and inform its beliefs as the search evolves. Secondly, the Gaussian beliefs were truncated from below at the admissible f value and above at three standard deviations.

Algorithm 3: Data Collection

1 Pick a set of training problem instances $\mathcal{T}$
2 Pick a search algorithm $\mathcal{S}$
3 **for** t *in* $\mathcal{T}$ **do**
4 Solve t with $\mathcal{S}$, while recording all expanded states
5 **for** *state* s *expanded by* $\mathcal{S}$ **do**
6 Solve s optimally
7 Store pair $(h(s), h\ (s))$

Data-Driven Nancy

The assumption-based instantiation of Nancy makes use of several assumptions about heuristic behavior. We next 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.

Data Generation The purpose of Nancy's assumptions is to obtain a better estimate of the possible true goal distances $h\ (s)$ 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 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 $h^0 \leq h$ for which we have a distribution in the data set. The distribution for h is extrapolated by shifting the distribution of h^0 by the difference between h and h^0, i.e., adding $(h - h^0)$ to each data point in the distribution of h^0. 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

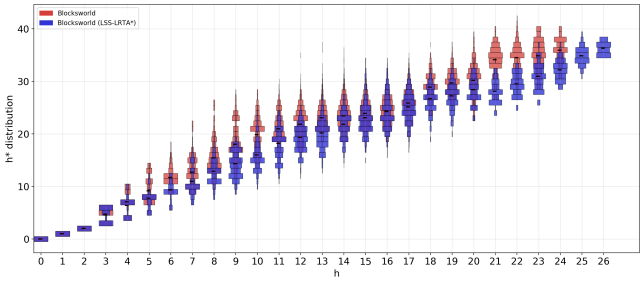

Figure 2: Beliefs in Blocksworld generated with weighted A* and LSS-LRTA*.

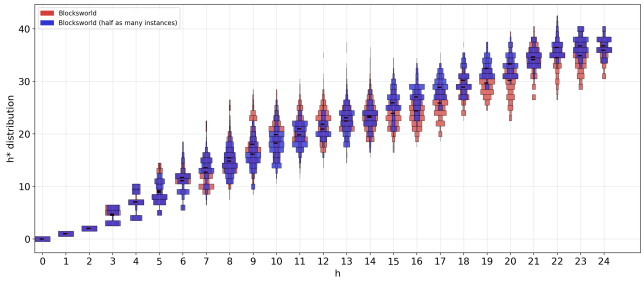

Figure 3: Beliefs in Blocksworld generated on 35 examples instances and on 17.

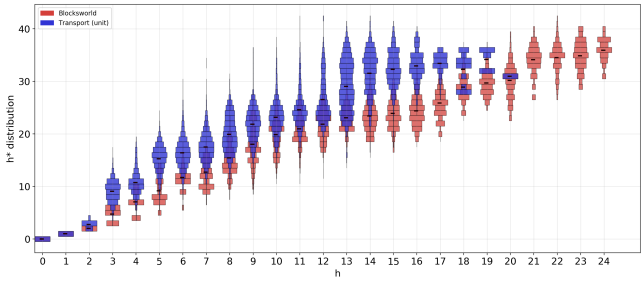

Figure 4: Beliefs generated for Blocksworld and Transport (unit-cost).

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 (e.g. Blocksworld with more than 10 blocks), 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. 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, the heuristic behavior can vary a lot between different domains, and domain-specific data can capture the behavior more accurately.

Since DDNancy is a suboptimal search algorithm, we also use a suboptimal search algorithm to sample the states for the data generation. Figure 2 shows the belief distributions that result from using weighted A* with a weight of 2 and LSS-LRTA* for the sampling. The generated beliefs 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 experiments in the later section, we use weighted A* with a weight of 2 to sample the training states.

A further key concern for any data-driven algorithm is how to determine the number of examples necessary to ensure that the data observed in those examples are representative for the general case. This is of special importance with our setup, since we only consider a subset of all states for any given training instance. Figure 3 shows a comparison of the resulting beliefs on Blocksworld when only using half of the available training instances as an empirical indication whether our data is sufficient. As expected, the spread of h values is smaller when reducing the number of sample states. Overall however, the distributions have similar shapes.

Figure 4 shows an example of the generated data for Blocksworld and the unit-cost version of Transport to demonstrate the differences in heuristic behavior on these domains. 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.

A similar comparison of the generated data for the classic search domains is shown in Figure 5. While the distributions are very smooth for the pancake puzzle and the racetrack domain, the expected value of the distributions on the 15-puzzle makes a large jump at $h = 4$. This shows the potential inaccuracy of the Manhattan distance heuristic in the 15-puzzle: there are states where the heuristic value is small, but an optimal solution still requires a significant number of moves.

Empirical Comparison

While Nancy is, to our knowledge, the first method for real-time heuristic graph search that bases its search strategy on

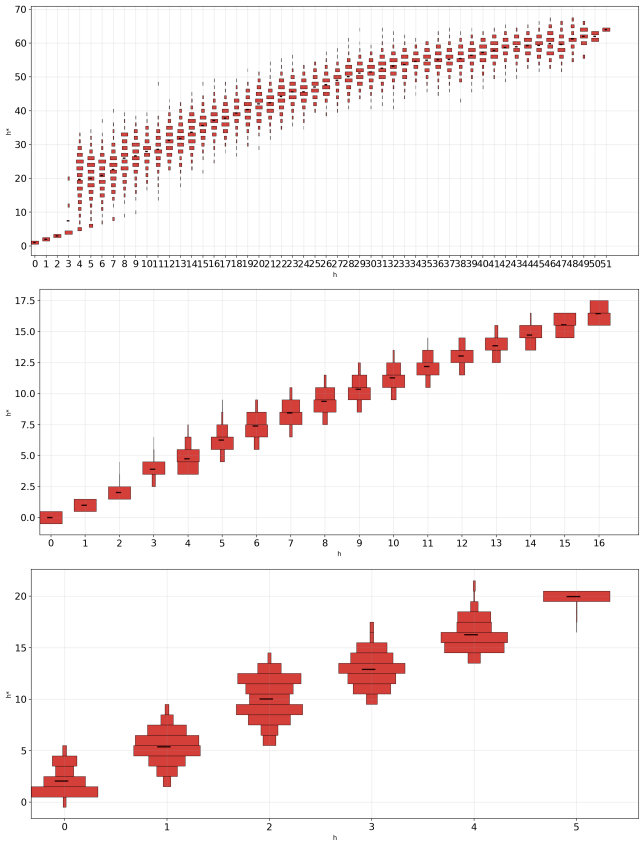

Figure 5: Beliefs gathered for the uniform-cost 15-puzzle (top), 16-pancake (middle), and Barto Racetrack (bottom).

belief distributions, there has been much previous work in the RL community on search methods that attempt to estimate and exploit value uncertainty. However, most RL domains feature stochastic actions, while in our setting the uncertainty stems entirely from the bounded computation of the agent. We consider two prominent approaches: interval estimation and Monte-Carlo tree search. In each case, we adapt the previous work to our setting and empirically compare it to Nancy.

Domains

We show experiments on the three classic search domains. First is the classic 100 15-puzzle instances published by Korf (1985). We test two variants: uniform-cost, in which every actions costs one, and heavy, in which the action cost is equal to the label of the moved tile. We use the Manhattan distance heuristic for all the real-time search algorithms.

Second is the pancake problem (Kleitman et al. 1975; Gates and Papadimitriou 1979; Heydari and 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 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.

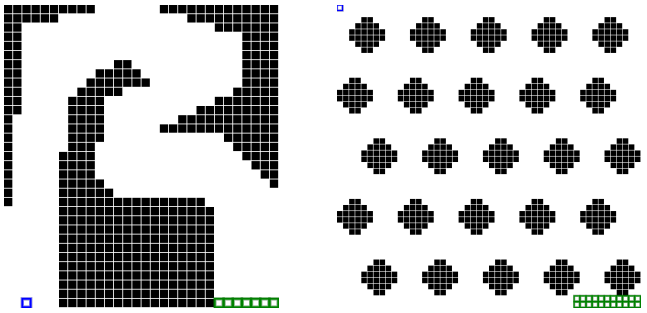

Figure 6: Barto (left) and uniform (right) Racetrack variants.

Third is the Racetrack domain which 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, and Singh 1995). Figure 6 shows the two maps used in our experiments. The track shown on the left was created by Hansen and Zilberstein (2001), and the cluttered track on the right by Cserna et al. (2018). The agent moves in a grid attempting to reach one of a set of goal locations while avoiding static obstacles. Each action 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 in that dimension. This is admissible. 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.

Interval Estimation

Interval Estimation (IE) (Kaelbling 1993; Strehl and Littman 2004) applies the philosophy of 'optimism in the face of uncertainty' to the problem of action selection in a two-armed bandit problem. While the method has previously been investigated primarily for use in MDPs and RL, we adapt it here for use in real-time deterministic planning. When deciding under which TLA the next node should be expanded, given the belief distributions of all the TLAs, 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. To adapt IE to the real-time search setting, we need to augment it with a mechanism for heuristic value updates. For each node in the LSS, we back up the belief from the child with the lowest lower confidence bound. Thus, the best frontier distributions under each TLA are eventually backed up to the TLA. The interval estimation approach naturally practices the spirit of uncertainty-based exploration in a very computationally efficient way.

Figure 7 shows an experimental comparison 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 perhaps marginally better on Racetrack. Overall, IE is very competitive, and yet simple to implement and computationally efficient.

Monte Carlo Tree Search

Monte-Carlo tree search (Browne et al. 2012) approaches such as UCT (Kocsis and Szepesvári 2006) are popular for solving stochastic problems such as with MDPs (Keller and Eyerich 2012) and POMDPs (Silver and 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). 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. 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 (without the preferred operators enhancement that is specific to domain-independent planning). 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 7. 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. Overall, the uncertainty-aware algorithms (Nancy, IE, and THTS) surpass the conventional LSS-LRTA* baseline, with Nancy and IE being the most robust.

Discussion

Viewed broadly, reinforcement learning considers how action selection should be informed by data that is gathered during execution. This is exactly what heuristic search strategies do. The states and costs computed during lookahead are data that inform action selection. As Nancy shows, a heuristic search can use this data in two ways. Clearly, the computed lookahead states in a real-time search setting inform the selection of the action for the agent to execute. But more broadly, the problem of designing any heuristic search strategy, even an off-line one, is an RL problem at the computational level, in that the search space computed so far can inform the choice of which nodes should be expanded next. For an optimal off-line search like A*, all nodes whose f values are less than the optimal solution cost C must be expanded, so there is little flexibility and less need for a sophisticated expansion strategy. But, in contrast, the tight resource limitations of real-time search strongly highlight the need for care in selecting even computational expansion actions.

This 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 node expansion gathers data that rule out those state space graphs that are inconsistent with the computed successor nodes, action costs, and h values. A goal state 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. Solving this 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. The data-driven version of Nancy highlights this. However, while Nancy does try to predict how its beliefs will change with additional search under frontier nodes, note that it is myopic and does not really plan at the metalevel. For example, if nodes with $h = 3$ were to be predicted to have higher expected distance to goal than nodes with $h = 4$, due perhaps to a misleading heuristic, data-driven Nancy will not realize that it must nonetheless ignore the tempting $h = 4$ nodes from time to time and try expanding some $h = 3$ nodes in order to eventually reach some $h = 2$ nodes and eventually the goal.

In related work, Lin et al. (2015) formalize the problem of metareasoning for an MDP and find that its computational complexity is polynomial in the time required to solve the MDP itself. This indicates the impracticality of optimal metareasoning, motivating approximations such as those used by Nancy.

Conclusion

Inspired by distributional methods from RL, the Nancy framework reconsiders real-time search as a decision making process where limited information creates uncertainty. Nancy models this uncertainty using belief distributions and reasons about it to guide the search. In this paper, we presented further experimental results regarding this approach. First, we presented detailed results regarding heuristic error data, which Nancy can use as the basis for its beliefs. Second, 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. This work illustrates how distributional methods from RL can be adapted to deterministic planning settings, and how

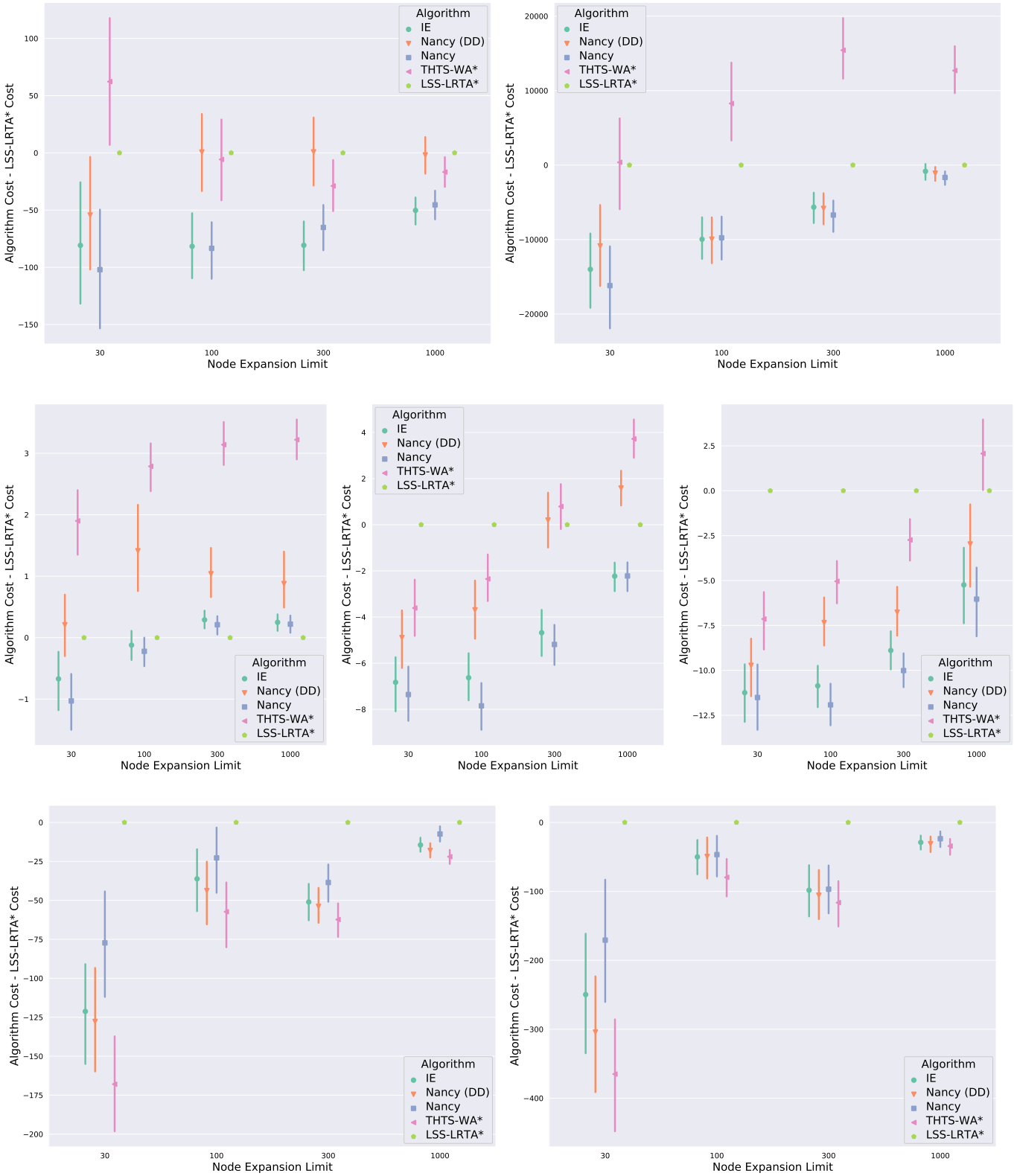

Figure 7: Comparison to IE and THTS. Top: 15-puzzle (left: unit, right: heavy); Middle: pancake (16, 32, and 40); Bottom: Racetrack (left: Barto, right: uniform).