Real-Time Planning as Decision-Making Under Uncertainty

Andrew Mitchell[1], Wheeler Ruml[1], Fabian Spaniol[2], Jörg Hoffmann[2], Marek Petrik[1]
[1]Department of Computer Science, University of New Hampshire, USA
[2]Department of Computer Science, Saarland University, Germany
ajx256@wildcats.unh.edu, ruml@cs.unh.edu, s9faspan@stud.uni-saarland.de,
hoffmann@cs.uni-saarland.de, mpetrik@cs.unh.edu

Abstract

In real-time planning, an agent must select the next action to take within a fixed time bound. Many popular real-time heuristic search methods approach this by expanding nodes using time-limited A* and selecting the action leading toward the frontier node with the lowest f value. In this paper, we reconsider real-time planning as a problem of decision-making under uncertainty. We propose treating heuristic values as uncertain evidence and we explore several backup methods for aggregating this evidence. We then propose a novel lookahead strategy that expands nodes to minimize risk, the expected regret in case a non-optimal action is chosen. We evaluate these methods in a simple synthetic benchmark and the sliding tile puzzle and find that they outperform previous methods. This work illustrates how uncertainty can arise even when solving deterministic planning problems, due to the inherent ignorance of time-limited search algorithms about those portions of the state space that they have not computed, and how an agent can benefit from explicitly metareasoning about this uncertainty.

Introduction

In some AI applications, such as user interfaces or fixed-wing aircraft control, it is undesirable for the system to exhibit unbounded pauses between actions. In real-time planning, an agent is required to select its next action within a fixed time bound. The agent attempts to minimize its total trajectory cost as it incrementally plans toward a goal. Many real-time heuristic search methods have been developed for this problem setting. Many of them 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 to form a local 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 the agent would incur if it selected each possible currently-applicable action, and then 3) commit to the lowest cost action and, 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 one is performed using A* (Hart, Nilsson, and Raphael 1968), the value estimates in step two are implicitly calculated for each node as the minimum f value among its successors (the 'minimin' backup), and the learning in step three is performed by updating h values for all nodes in the LSS using a variant of Dijkstra's algorithm.

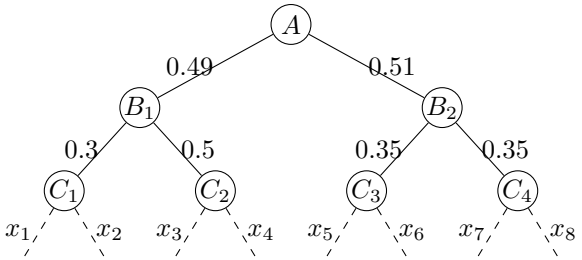

Figure 1: Should an agent at A move to B_1 or B_2?

However, this paradigm is not necessarily optimal. This becomes apparent when one considers the problem from the perspective of decision-making under uncertainty. For example, Pemberton and Korf (1994) observed that, given an LSS, the optimal decision for the agent is not, in general, to head toward the frontier node with the lowest f value. Figure 1 shows their example LSS, in which an agent located at node A must decide whether to transition to node B_1 or B_2. All nodes at depth three after the C_i are goals. Recall that the principle of rationality demands that an agent take an action that minimizes expected cost. The edge costs x_i are unknown but uniformly distributed between 0 and 1, so $h = 0$ for all nodes C_i. C_1 is the node of lowest f (namely 0.79) so a typical real-time search would move to B_1. However, moving to B_2 minimizes the agent's expected cost, because in the next iteration the costs $x_5, ..., x_8$ will be revealed and the expected minimum of these four values (0.2) yields a total expected cost for B_2 (1.06) less than that of B_1 (1.066, see Pemberton, 1995, eq. 1 for details). 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, it was noted by Mutchler (1986) that A*'s

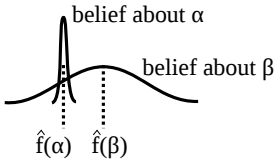

Figure 2: Should an agent expand nodes under α or β?

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 about which currently-applicable action is best, it is sometimes beneficial to gain knowledge by expanding a different node. For a simple example, consider the situation depicted in Figure 2, in which we represent the agent's current belief about the remaining plan cost that will be incurred by committing to an action by a probability distribution over possible costs. The expected value is denoted by $\hat{f}(\cdot)$. If the agent is quite certain about the value of action α but quite uncertain about the value of a different action β, then expanding frontier nodes under α can be less useful than expanding under β, even if β is currently believed to have a higher expected cost. In other words, 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 well-understood good-looking option. This problem does not arise in off-line optimal search, in which every node whose f value is less than the optimal solution cost must be expanded.

Although these insights seem powerful, they have not yet resulted in generally-applicable real-time search algorithms. In this paper, we advance toward practical algorithms that realize the reasoning under uncertainty perspective. We study strategies for expanding nodes and methods for backing up their information for decision making, culminating in a novel general-purpose method for real-time search that we call Nancy. Experimental results in random trees and in the sliding tile puzzle suggest that Nancy makes better use of limited node expansions than conventional real-time search algorithms. While additional work will be required to engineer an optimized implementation, Nancy already shows how reasoning about uncertainty can play an important role in resource-constrained search even for deterministic domains.

Previous Work

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 (see also Schulte and Keller, 2014), 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 (Bellemare, Dabney, and Munos 2017, and references therein).

Decision-making from Lookahead

Real-time planning requires us to commit to one of the actions applicable at the agent's current state — we call these top-level actions (TLAs). The assumption behind lookahead is that the f values of frontier nodes are more informed than those of the current state's immediate successors. Our belief about the total trajectory cost we will be able to achieve by moving through a node n depends on our beliefs about n's successors. Therefore, after generating an LSS, we want to somehow propagate the frontier f values back up the tree through their ancestors to estimate the value of each TLA. In this section, we identify and explore four distinct backup rules, each with its own assumptions about the unexplored portion of the state space and the behavior of the agent: minimin, Bellman, Nancy, and Cserna. Using these, we will also see how to generalize Pemberton (1995)'s k-best method to arbitrary trees.

Minimin Backups

Minimin has been used in real-time search since at least the work of Korf (1990). It assigns a parent node p the minimum f among its successors $S(p)$:

$$f(p) = \min_{c \in S(p)} f(c)$$

The best path to a goal must go through a successor, so the parent's f value must be at least as large as its best child's. Minimin is used implicitly in the learning phase of LSS-LRTA*, as adjusting states' h values so as to satisfy the minimin equation raises them as high as possible while preserving admissibility (Koenig and Sun 2008). While this is a desirable property for h learning, we noted earlier that f is a lower bound rather than an expected cost, and is thus not appropriate as a basis for rational action selection. Minimin becomes rational and optimal when there is no uncertainty and the frontier f values are equal to the true f^* values. While this case is sufficiently approximated by the end of an A* search, it does not accurately model real-time planning.

Bellman Backups

A Bellman backup (Bellman 1957) explicitly estimates expected value, again as the minimum over successors:

$$\hat{f}(p) = \min_{c \in S(p)} \hat{f}(c)$$

By using expected value, the Bellman backup recognizes the uncertainty that remains between states at the search frontier and any goals lying beyond. It is used as the basis of action selection by Pemberton and Korf (1994) and for both action selection and node expansion by Kiesel, Burns, and Ruml (2015). (In its usual form with stochastic transitions, it is of course also the basis for value iteration methods for MDPs.)

The Bellman backup has two weaknesses. First, it conveys only a scalar expected value. While this is all that is needed for selecting a TLA, we saw in Figure 2 and will see again below that having a complete belief distribution available is useful both for more sophisticated backups and for guiding lookahead expansions. The second weakness is that it assumes that no additional information will become available as the agent traverses the LSS. In real-time search, by the time the agent reaches a node deeper in the tree, further lookahead will have produced additional information that can inform its choices. In Figure 1, for example, Bellman will backup the expected value of C_3 (or C_4) to B_2, ignoring the fact that by the time the agent makes a decision at B_2, it will have the best of $x_5, ..., x_8$ to choose from, rather than just the values below C_3 (or C_4). Backing up from merely one child does not capture what the agent can expect to achieve. This also raises a subtle point that, to our knowledge, has not been addressed in previous work: there is a distinction between the lowest cost of any solution through a node $f^*(n)$, which is independent of the agent's resources and planning algorithm, and the cost that a resource-bounded agent can actually expect to achieve by moving through n, which we notate $f^@$. The latter depends on many factors, including lookahead budget, expansion strategy, and backup rule, and will likely be higher than f^* unless we have full knowledge of the relevant remaining state space. While off-line planners compute f^*, we need to take the agent's information state into account and aim to estimate $f^@$.

Nancy Backups

To address the first weakness of Bellman backups, lack of a full belief distribution, we introduce the Nancy backup, which assigns to the parent the belief of the child with the lowest expected cost:

$$B(p) \sim \operatorname*{argmin}_{d \in \{B(c) | c \in S(p)\}} \mathbb{E}(d)$$

Of course, this requires that we define the agent's belief about $f^@$ at every frontier node. This is not hard (O'Ceallaigh and Ruml 2015) and we will see examples below. While Nancy backups are very similar to Bellman backups, we will see below that having full belief distributions can yield advantages. They do share Bellman's weakness of assuming that no more information will become available. As such, they are only correct if this assumption holds or if the best frontier node under a TLA is so much better than all of the others that its belief is disjoint from theirs.

Cserna Backups

The second weakness of Bellman backups for real-time search, insensitivity to additional information, was pointed out by Cserna, Ruml, and Frank (2017). Although they did not formulate it explicitly, their work implies a new backup rule that assumes that, by the time the agent reaches a node, it will know the true value of each successor and be able to choose the minimum among them. Thus the probability we assign to a value for the parent corresponds to the probability that it will be the minimum of all the successor's values.

As a CDF, this is

$$\mathbb{P}\left[f^{@}(p) \leq x\right] = \mathbb{P}\left[(\min_{c \in S(p)} f^{@}(c)) \leq x\right] \quad (1)$$

We assume that the probability of a configuration of values for the children follows our current beliefs about them and, in our implementation, that the beliefs of the successors are independent. To compute the Cserna backup involving two children A and B, our implementation considers all possible pairs of values a and b and adds their minimum, weighted by the probability of getting that pair, to the belief distribution of the parent.

We note that the Cserna backup is associative — performing a Cserna backup directly on all the frontier nodes beneath a TLA will result in the same belief as performing Cserna backups recursively up the tree from the frontier to the TLA:

$$\begin{aligned}\mathbb{P}\left[f^{@}(p) \leq x\right] &= \mathbb{P}\left[\min_{c \in S(p)} \min_{d \in S(c)} f^{@}(d) \leq x\right] \\ &= \mathbb{P}\left[\min_{d \in S(S(p))} f^{@}(d) \leq x\right] \\ &= \mathbb{P}\left[\min_{d \in S(\ldots S(p))} f^{@}(d) \leq x\right]\end{aligned}$$

where the set-valued S is defined as $S(A) = \bigcup_{a \in A} S(a)$.

Because they take into account the distributions of all children under a node, Cserna backups will make the correct decision in Figure 1, moving to B_2 because of the high chance of a low value among the $x_5, ..., x_8$ rather than to B_1 toward the frontier node with minimum f. In other words, because the expected minimum can be lower than the minimum of the expectations, the expected value of a Cserna backup can be lower than the expected value of the best child distribution, which can lead to different action selection than Bellman backups.

Cserna backups assume that, by the time we reach a node, the true value of every child will be available, so that we can choose the minimum cost child. However, in the case of real-time search, this assumption only holds if the lookahead is large enough to exhaust the remaining reachable state space on the next iteration. Otherwise, Cserna backups will be optimistic about a node's value.

The k-Best Strategy

With these backup strategies defined, it becomes straightforward to generalize Pemberton (1995)'s k-best method to arbitrary trees. His closed-form analysis of the last incremental decision problem can be seen as a specialization of Cserna backups to his random trees. We will define the k 'best' frontier nodes under a TLA as those having the lowest $\hat{f}$ values. We backup all frontier nodes using Nancy backups, except when two or more of the best meet at the same parent, in which case we use a Cserna backup on them. We now consider this parent one of the 'best' and continue up the tree. This method interpolates between Nancy backups ($k = 1$) and Cserna backups ($k \geq$ maximum number of frontier nodes under any TLA) while limiting the number of expensive Cserna backups to at most $k - 1$ under each TLA.

Experiments

While each of the backup strategies has cases in which it is optimal, none of these correspond to actual real-time search. We now experimentally evaluate the backup rules to see which perform best in practice. We use as benchmarks both random trees (following Pemberton and Korf (1994)) and the classic 15 puzzle benchmark (following Korf (1990)). The random trees were generated lazily yet deterministically, with branching factor 2 and edge costs uniformly distributed between 0 and 1 (so $h = 0$ for all nodes). The g cost of a node is equal to the sum of the edge costs taken to reach it, and every leaf is a goal. We used the 100 random 15 puzzles first generated by Korf (1985) and the Manhattan distance heuristic. We implement beliefs numerically as discrete distributions at 100 evenly spaced values. Cserna backups use numerical integration, collapsing nearest values when necessary to keep the size at 100.

Our first experiment tests the same last incremental decision problem studied by Pemberton (1995): the LSS is the first 9 levels of a depth 10 tree. After committing to its first action, the agent observes the last level and follows an optimal path. We would expect Cserna backups to be optimal in this setting. We used 30,000 random trees. For minimin, the frontier values are the f costs, which equal the g costs. For Bellman, we estimate $\hat{f}(n)$ as $g(n) + 0.23d(n)$, where $d(n)$ is an estimate of the number of actions from n to a goal (equal to $h(n)$ in domains with unit edge costs) and 0.23 is an estimate of h's per-step error (obtained from pilot experiments on random trees of depth 100, where the average solution cost was about 23). For Nancy and Cserna, we tried both the correct beliefs, derived via Cserna backups from [0, 1] uniform distributions at the lookahead frontier, and more general approximate beliefs represented by Gaussians. The Gaussian beliefs are centered on $\hat{f}$ and, following O'Ceallaigh and Ruml (2015), have variance proportional to the difference between a node's $\hat{f}$ and f values:

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

Note that, because the difference between $f(n)$ and $\hat{f}(n)$ is proportional to $d(n)$ (the distance from the goal), this reflects the common assumption that heuristics are more accurate as one approaches a goal. The beliefs were implemented as truncated Gaussians bounded from below at the admissible f value and above at three standard deviations.

Figure 3 shows the mean solution cost achieved by the various backup rules, relative to Cserna with correct beliefs, along with 95% confidence intervals. The top-to-bottom order in the legend corresponds to the left-to-right order in the plot. The results confirm our expectations that Cserna is optimal for the last incremental decision problem, that 1-best is equivalent to minimin, that k-best converges to Cserna as k increases, and that Nancy is equivalent to Bellman. They also indicate that Gaussian beliefs behave reasonably, leading to only a small increase in cost compared with knowing the correct belief distribution.

However, most decisions in real-time search occur far from goals. We next test the strategies in the context of lim-

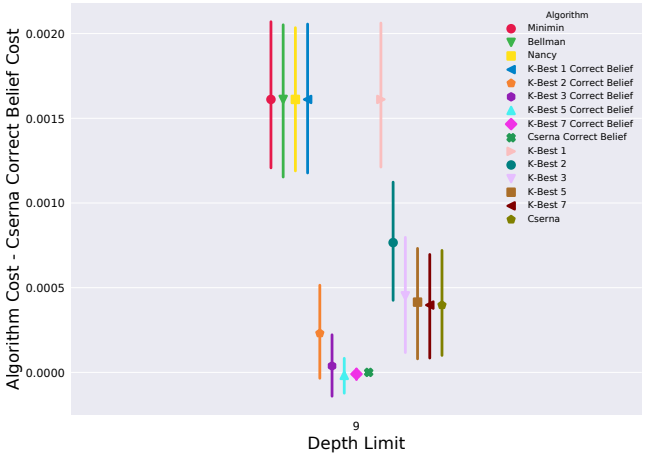

Figure 3: Backup rules on random trees with one level unknown.

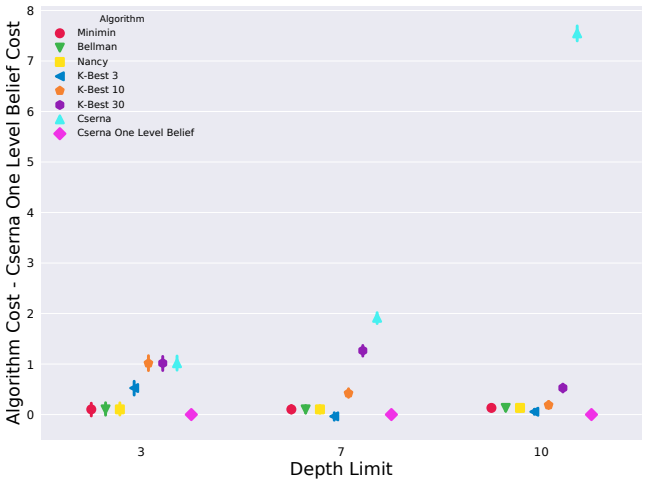

Figure 4: Backup rules with depth-first lookahead on random trees.

ited lookahead in 1,000 trees of depth 100 (receding horizon control). We used both general Gaussian beliefs and, following Pemberton (1995), beliefs that assume only one level of the tree remains below the frontier. Because the agent can make better decisions with larger lookaheads, we used different estimates for $\hat{f}$ based on the lookahead depth or node limit, derived from pilot experiments on random trees ($g(n) + c \cdot d(n)$ where c varied from 0.221 to 0.295). For the 15 puzzle, $\hat{f}$ was estimated using the on-line learning approach of Thayer, Dionne, and Ruml (2011) (the 'global average' one-step error model for h and d). Figure 4 plots relative solution cost as a function of the lookahead depth for a simple bounded depth-first lookahead. Minimin, Bellman, and Nancy perform the same, as expected. Interestingly, Cserna with Gaussian beliefs performs very poorly, even though the one-level beliefs are no longer correct.

Finally, Figure 5 uses A* lookahead, as used in real-time search algorithms such as LSS-LRTA*. In this case, nodes

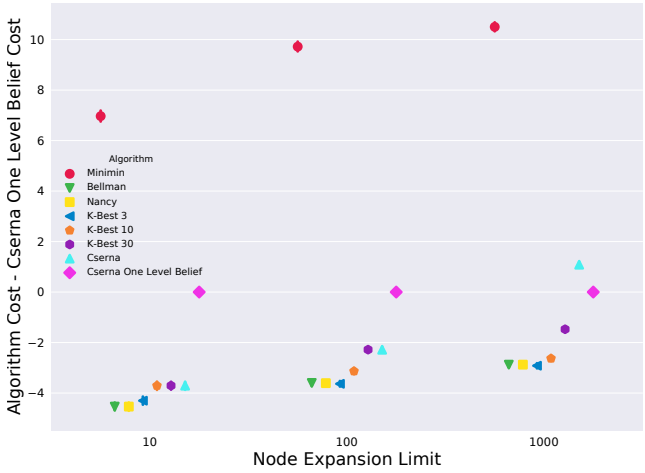

Figure 5: Backup rules with A* lookahead on random trees.

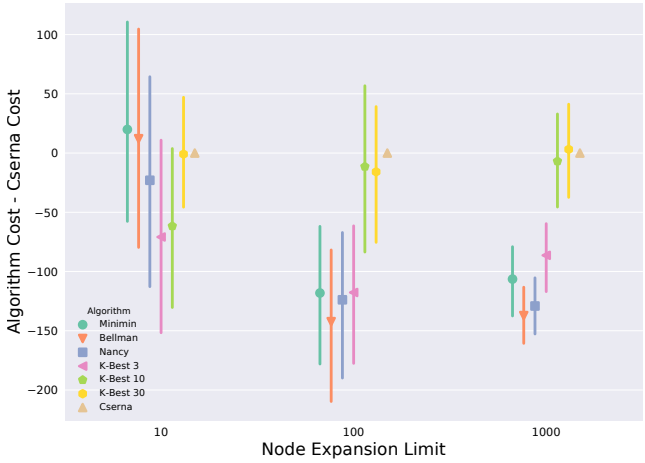

Figure 6: Backup rules with A* lookahead on 15 puzzles.

on the lookahead frontier can be at different depths. Because $h(n) = 0$ for all nodes, minimin is essentially backing up g values and performs very poorly, while Bellman and Nancy use more sensible expectations adjusted for depth and perform well. Gaussian beliefs outperform one-level beliefs at lower lookaheads. Figure 6 shows the corresponding results on the 15 puzzle. Because this state space is a graph and the agent can return to the same state, we incorporate the learning step of LSS-LRTA*, which updates h values on-line and is guaranteed complete. Overall, the results are remarkably similar to the random trees: Bellman and Nancy are better than minimin, and Cserna performs poorly. We conjecture that Cserna's assumption that each child's true value will be revealed is too strong, and that real search trees are better approximated by Bellman and Nancy's assumption that little new knowledge will be brought to bear.

To summarize, we have seen that minimin and Cserna backups perform poorly for real-time search in practice, while Bellman and Nancy seem to work well. As we turn our attention to node expansion strategies, we will use Nancy

backups, as they provide full belief distributions for the strategies to use.

Choosing Nodes to Expand

Given an agent's current beliefs about the $f^{@}$ values of the TLAs, it is not immediately obvious which frontier node to expand. Mutchler (1986) showed that the most common choice, the node with lowest f, is not optimal. Kiesel, Burns, and Ruml (2015) propose $\hat{f}$, but as Figure 2 showed, taking our uncertainty into account can be important. Because we are using Nancy backups, the only node whose expansion can change our belief about a TLA is the one with the minimum $\hat{f}$ under that TLA. We will consider several candidate criteria to use for choosing which node to expand, and we optimize them myopically. We predict, for a given TLA, how our belief about it would change if we were to expand the best frontier node under it. (We discuss one way of forming these predictions below.) We then evaluate the resulting belief in the context of the other TLAs' unchanged beliefs according to a desirability criterion, and choose the node whose expansion gives the most desirable set of beliefs. The criteria we consider are confidence, expected advantage, and risk. Because action selection will choose the TLA with the lowest expected value, we will call the current such TLA the 'best action' or α.

We define *confidence* as the probability, given a set of beliefs, that the best action actually has the lowest $f^{@}$ value. Said another way, this is the probability that, given samples from each TLA's belief, the one from α is lowest. By expanding nodes so as to maximize our confidence, we reduce the overlap among the leading beliefs, helping us be sure that we choose the best action. In Figure 2, for example, expanding under β may well increase confidence faster than expanding under α. While this seems sensible, note that confidence can have odd behavior. For example, an action β might have most of its belief's probability mass below the belief for α, making β very likely to be lower, yet an extreme outlier in β's distribution could cause its $\hat{f}$ to be higher than α's. The central weakness of confidence is that, as a probability, it does not take solution cost into account. Given two possible expansions that equally improve the probability that α is best, it cannot distinguish which might improve the agent's expected cost more.

To capture this, we define *expected advantage*, given a set of beliefs, as the difference between the expected costs of the best action $\hat{f}(\alpha)$ and the second best action $\hat{f}(\beta)$. Note that expanding nodes to maximize expected advantage is not the same as expanding the node with lowest $\hat{f}$ — it might be the case that expanding under β moves it further from α than expanding under α moves it away from β. There are two serious deficiencies in this criterion, however. First, it does not take into account the uncertainty of our beliefs and cannot distinguish between two expansions that move $\hat{f}$ values equally but reduce variance differently. Second, and related to this, in practice we will model belief change due to search as reduction in variance, leaving $\hat{f}$ unchanged. So a criterion insensitive to uncertainty is a non-starter.

Finally, we arrive at our final criterion, *risk*, which we define as the expected regret in those cases in which α was not the correct choice. For two TLAs, this is

$$\mathbb{E}\left[f^{@}(\alpha) - f^{@}(\beta) \;\middle|\; f^{@}(\beta) < f^{@}(\alpha)\right],$$

where $\mathbb{E}$ is the (conditional) expectation operator. This generalizes to additional TLAs by considering every outcome where one of the β_i is better than the current α. This is done using our current beliefs about the TLAs in concert with estimates about how each belief would change if we were to search under its corresponding best frontier node. In our implementation, risk is computed numerically by taking each combination of possible values for the best TLA, α, and the other TLAs, β_i, and finding $a - b_i$ where a is a possible $f^{@}$ value in α and b_i is a possible $f^{@}$ value in β_i where $b_i < a$, weighted by the probability of that combination. We do this risk computation once for each TLA, temporarily adjusting our belief about that TLA to be the predicted post-expansion belief. As explained further below, we model our predicted post-expansion belief about a TLA as having the same mean as the pre-expansion belief, but a smaller variance because variance is a function of heuristic error, which we assume will decrease as additional expansions bring the search closer to a goal. We then expand the frontier node under the TLA whose predicted post-expansion belief resulted in the least risk.

Experiments

We now turn to an experimental evaluation of node expansion strategies. To use risk-based expansion, we require a model of how beliefs change due to search. We use a variant of the scheme presented by Cserna, Ruml, and Frank (2017). Note that the variance of our beliefs is based on the distance to the goal from the frontier node from which we inherited our belief ($\hat{f} - f$, Eq. 2). Expanding the node will reduce our distance to goal, thus we reduce its variance by $\min(1, d_s/d(n))$ (Eq. 2 of Cserna, Ruml, and Frank (2017)), where $d(n)$ is the estimated number of steps to the goal and d_s is the *expansion delay* (a measure of search vacillation equal to the average number of other nodes generated between when a node is generated and when it is expanded). We assume the expected value remains unchanged (its expected behavior).

We test expansion strategies in the context of Nancy backups, given their success in the experiments above. Frontier nodes are given Gaussian beliefs. We compare risk-based expansion with conventional lookahead techniques, including breadth-first, A*, and $\hat{f}$-based expansion. Figure 7 shows mean solution cost relative to A* on random trees of depth 100. We see that, for lookaheads of more than 3 nodes, breadth-first is dominated by the more flexible strategies. $\hat{f}$ lookahead is often better than A*'s f-based strategy, confirming the results of Kiesel, Burns, and Ruml (2015). But risk-based expansion is clearly the best, showing strong benefits over the other strategies for lookaheads of 10 nodes and greater.

Figure 8 presents the corresponding results from 15 puzzles. Breadth-first is again poor, but $\hat{f}$ is not clearly supe-

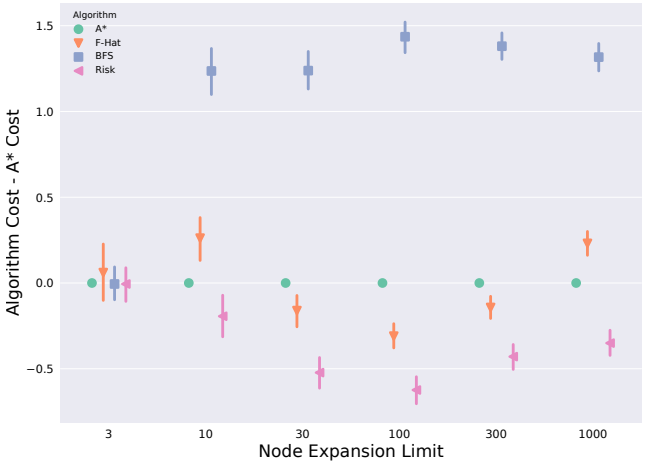

Figure 7: Expansion strategies on random trees.

rior to f. However, aside from poor performance with a 3-node lookahead, risk-guided expansion again seems superior. For comparison, we also show the performance of LSS-LRTA*, a popular real-time search method that uses A* expansion, minimin backups, and moves toward the frontier node of lowest f. The Nancy algorithm of risk-based expansion, Nancy backups, and $\hat{f}$ action selection offers significantly better performance at lookaheads of 30 and higher. Although it may appear that risk is converging back toward A* as lookahead increases, this is because both algorithms are approaching optimal solutions. Figure 9 shows the average optimality gap (cost over optimal) of each method as a percentage of A*'s average gap. The results show that risk has 65% of A*'s difference from optimal at a lookahead of 1000, and even less at 300.

To summarize, we have seen that a lookahead strategy based on minimizing risk can outperform traditional search strategies. By using the belief distributions provided by Nancy backups, it can determine when it may be beneficial to expand nodes other than those having the best expected value.

Discussion

While our experimental results are promising, it is important to recognize that the methods were compared using a fixed number of node expansions rather than CPU time. Although Nancy backups are not particularly expensive, our implementation of risk-based expansion numerically manipulates distributions and integrates over them. Such metareasoning overhead can be amortized by performing several expansions per decision, but it would be useful to engineer an optimized method. We believe that a lightweight approximation of risk-based expansion should be possible, perhaps along the lines of Thomspson sampling (Thompson 1933) or UCB (Auer, Cesa-Bianchi, and Fischer 2002).

Although we successfully adapted previous work on belief distributions for heuristic search, the issues of how best to represent beliefs and how to acquire them, perhaps through on-line learning, deserve sustained attention.

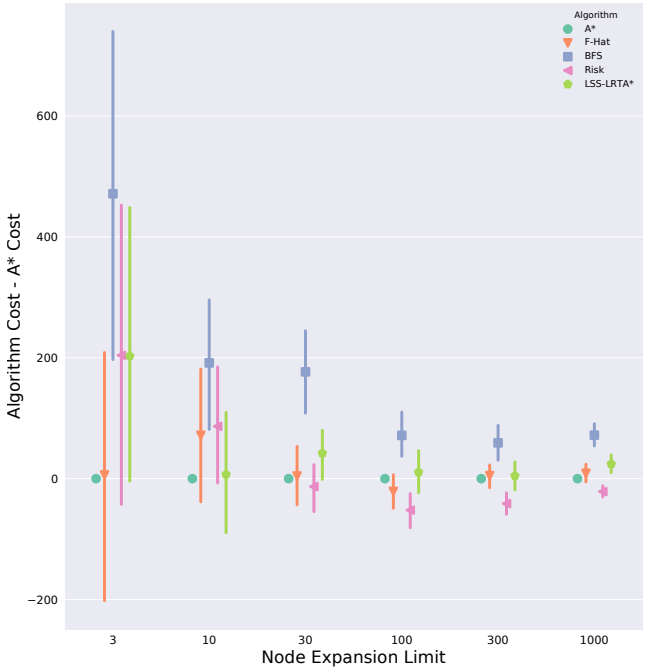

Figure 8: Expansion strategies on 15 puzzles.

We explored a greedy one-step lookahead approach to minimizing risk during node expansion, but it would be interesting to explore the possible benefit of more elaborate schemes.

Conclusion

It is tempting to believe that, although reasoning under uncertainty is a popular topic in research on MDPs and POMDPs, it is not relevant for search in deterministic domains. However, this work illustrates how uncertainty can arise even in deterministic problems due to the inherent ignorance of search algorithms about those portions of the state space that they have not computed. Rationality demands that an agent select an action that minimizes expected cost, but this leaves open how to select nodes for expansion and how to aggregate frontier information for decision-making. We examined four backup rules, including two new ones, and a new expansion strategy that attempts to minimize risk. Experimental results on random trees and the sliding tile puzzle suggest that the new Nancy method provides results superior to traditional approaches. This work shows how an agent in a general real-time search setting can benefit from explicitly metareasoning about its uncertainty.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A distributional perspective on reinforcement learning. In *Pro-*

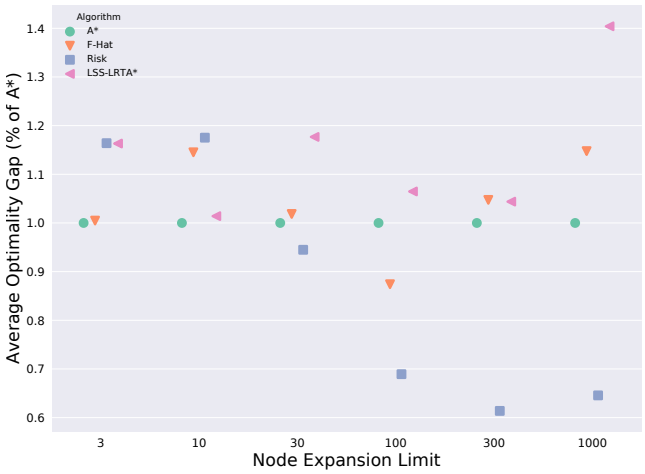

Figure 9: Optimality gap of expansion strategies on 15 puzzles as a percentage of A*'s.

ceedings of the Thirty-Fourth International Conference on Machine Learning (ICML-17).

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

Cserna, B.; Ruml, W.; and Frank, J. 2017. Planning time to think: Metareasoning for on-line planning with durative actions. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-17)*.

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

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

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

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

Koenig, S., and 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. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

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

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

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

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

Pemberton, J. C., and 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)*.

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)*.

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

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

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

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25(3/4):285–294.

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