Metareasoning in Real-Time Heuristic Search

Dylan O'Ceallaigh and **Wheeler Ruml**
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
dylan.oceallaigh at gmail.com, ruml at cs.unh.edu

Abstract

Real-time heuristic search addresses the setting in which planning and acting can proceed concurrently. We explore the use of metareasoning at two decision points within a real-time heuristic search. First, if the domain has an 'identity action' that allows the agent to remain in the same state and deliberate further, when should this action be taken? Second, given a partial plan that extends to the lookahead frontier, to how many actions should the agent commit? We show that considering these decisions carefully can reduce the agent's total time taken to arrive at a goal in several benchmark domains, relative to the current state-of-the-art. The resulting algorithm can dynamically adjust the way it interleaves planning and acting, between greedy hill-climbing and A*, depending on the problem instance.

Introduction

Real-time heuristic search refers to the setting in which a search must finish within a fixed amount of time. Because it may be impossible to find a complete path from the start state to a goal within this time, the search is only required to return the next action for the agent to take. The search is therefore iterated until a goal is reached. Because execution begins before the optimality of the path is verified, no real-time search can guarantee that the agent will follow an optimal trajectory. However, real-time search can nicely handle the realistic setting in which search and execution can happen in parallel: while the agent transitions from state s_i to s_j, the search can plan which action to take from s_j, where the duration of the transition is used as the bound on planning time. This is the setting we address in this paper.

Because we are addressing concurrent planning and execution, we will use goal achievement time (GAT) (Hernández et al. 2012; Burns, Kiesel, and Ruml 2013) as our main evaluation metric. This is the total time taken from the start of the first search iteration to the arrival of the agent at a goal. For an offline algorithm such as A*, this is the sum of planning and execution time. A* will spend a lot of time searching, but then the shortest possible time executing. Because of its real-time constraint, real-time search may execute a longer plan than A*, but because most of its planning will happen concurrently with execution, it may have a shorter GAT than A*. If the cost of an action equals the time taken to execute it, then a real-time search for minimizing GAT can be thought of as minimizing total cost, as usual, as long as the cost of identity actions are taken into account.

Because each search iteration is taking place under a time constraint, a real-time search must be careful about its use of computation time. After performing some amount of lookahead, a real-time search will have identified a most promising state on the search frontier, along with the path leading to it from the current state. It then faces two fundamental questions: 1) should it continue to search, gaining increased confidence that it has identified the correct action to take, or is it ready to begin executing the action that currently appears to be best? In many domains, the agent is allowed to idle in the current state, effectively executing a 'identity' or 'no-op' action while performing additional search. While this deliberation will delay the agent's arrival at a goal, the delay may be worthwhile if the additional planning allows the agent to avoid selecting a poor action. This is perhaps the most fundamental question an agent faces when planning and acting are allowed to interleave or run concurrently. And 2) if the algorithm decides to act, should it commit to all the actions leading to the most promising frontier node, or just some prefix?

Most existing real-time search algorithms decide these questions at design time. For example, the seminal RTA* algorithm of Korf (1990) always commits to a single action, while state-of-the-art algorithms like LSS-LRTA* (Koenig and Sun 2009) and Dynamic $\hat{f}$ (Burns, Kiesel, and Ruml 2013) always commit to the entire path to the frontier. None of these algorithms take advantage of identity actions. In this paper, we explore whether it can be beneficial to make these decisions dynamically during planning. Considering identity actions, for example, allows a real-time search to decide at runtime whether to behave more like greedy hill-climbing, and commit to an action after only limited lookahead, or behave more like A* (Hart, Nilsson, and Raphael 1968), and explore all the way to a goal before starting to act.

We view our approach as a form of metareasoning, in which the search algorithm governs its own behavior by attempting to estimate the effects of committing to actions versus doing more search. Because the effects of search are nec-

1. until a goal is reached
2. perform a best-first search on $\hat{f}$ until *time bound*
3. update heuristic values of nodes in CLOSED
4. $s \leftarrow$ state in OPEN with the lowest $\hat{f}$
5. start executing path to s
6. *time bound* $\leftarrow$ execution time to reach s
7. *OPEN* $\leftarrow \{s\}$; clear CLOSED

Figure 1: Pseudocode for the Dynamic $\hat{f}$ algorithm.

essarily uncertain, the challenge is to efficiently estimate the probability of various outcomes. We believe that the metareasoning perspective provides an elegant and principled way to address the fundamental trade-off between search and execution in planning and combinatorial search.

Concretely, the paper proposes two extensions of the Dynamic $\hat{f}$ algorithm, one addressing identity actions and one addressing prefix commitment. We carefully examine their behavior on a new set of handcrafted pathfinding problems and then run large-scale tests on four benchmark domains. We find that the new techniques successfully allow a real-time search to adapt its behavior between greed and deliberation, matching or outperforming previous state-of-the-art algorithms. This work represents a new state-of-the-art in the application of metareasoning to real-time search. We hope it spurs more investigation into the application of metareasoning in heuristic search.

Previous Work

LSS-LRTA*

Local Search Space-Learning Real-Time A* (LSS-LRTA*) is a state-of-the-art real-time search algorithm that has been recommended for situations in which there is an explicit real-time constraint per action (Koenig and Sun 2009). It consists of a two step process. First, LSS-LRTA* performs an A*-like search from the agent's current state toward a goal until some expansion limit is reached. Second, LSS-LRTA* selects the best state on the search frontier, queues the best path to that state for the agent to execute, much as A* does for the complete path to the goal, and proceeds with a Dijkstra-like backup process where the h value of each state in the lookahead search space is updated to the h value of its best child plus the edge cost between the two. This intensive learning step allows the algorithm to escape local minima much faster than previous real-time search algorithms.

Dynamic $\hat{f}$

Dynamic $\hat{f}$ makes two small modifications to LSS-LRTA*. A pseudocode sketch is shown in Figure 1. First, instead of using a fixed time bound, Dynamic $\hat{f}$ sets the time bound for search dynamically after each search iteration based on the duration of the actions that have been queued for execution (line 6). The longer it will take the agent to execute the actions that have been committed to, the more computation can be done by the search before it must terminate the subsequent iteration. This can result in a positive feedback loop of longer queued paths and yet more search time, allowing the search to perform larger learning steps and ultimately reaching the goal much faster. In contrast, LSS-LRTA* tries to minimize total search effort, and thus avoids search during execution. Under the GAT metric, concurrent search can be regarded as free.

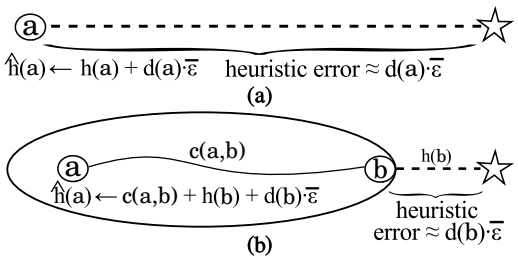

Figure 2: Backed up heuristic error in Dynamic $\hat{f}$.

The second modification that Dynamic $\hat{f}$ employs is an inadmissible heuristic, notated $\hat{h}$ (line 2). This value is our unbiased best guess about what the node's true f^* value is, rather than a lower bound. Just as $f(s) = g(s) + h(s)$, we will write $\hat{f} = g(s) + \hat{h}(s)$. In Dynamic $\hat{f}$, the search frontier is sorted on $\hat{f}$ instead of f. While any $\hat{h}$ could be used, in experiments report below, we use a version of the admissible h that is debiased online using the 'single-step error global average' method of Thayer, Dionne, and Ruml (2011). During search, the error ϵ in the admissible h is estimated at every expansion by the difference between the f value of the parent node and the f value of its best successor (these will be the same for a perfect h). If $\bar{\epsilon}$ is the average error over all expansions so far and $d(s)$ is an estimate of the remaining search distance (number of edges along the path) from a state s to the nearest goal, then $\hat{h}(s) = h(s) + \bar{\epsilon} \cdot d(s)$. Inadmissible heuristics have shown promise in other suboptimal search settings as well (Thayer 2012).

In Dynamic $\hat{f}$, the next action to take from the current state is the one leading to the successor with the best $\hat{f}$ value. As noted by Burns, Kiesel, and Ruml (2013), this requires some subtlety. If node a inherits its f value from a node b on the search frontier, then the heuristic learning step (line 3) will update its h value using the path cost between a and b (the difference in their g values) and b's h value. The remaining error in the estimate of a's f value will then derive from the error in b's h value. This is illustrated in Figure 2. To take this into account, we compute $\hat{h}(a) = h(a) + \bar{\epsilon} \cdot d(b)$. This means that, as we learn updated h values, we record the d values of the nodes they were inherited from.

Metareasoning

One of the first applications of metareasoning to heuristic search was Decision-Theoretic A* (DTA*) (Russell and Wefald 1991). DTA* emits actions individually, like a real-time search, but at each step it estimates how much search to perform. To do this, it weighs the time required to search against the likelihood that further search will change the selected action and the expected reduction in total GAT. If fur-

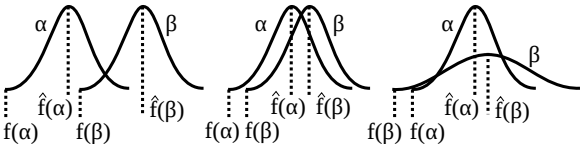

Figure 3: The three scenarios considered by Ms. A*.

ther search is predicted to reduce the expected GAT more than the time taken for the searching, then the algorithm implicitly executes an identity action and performs additional search. This approach is extremely elegant (and inspired the approach taken in this paper). However, to estimate the effects of further search, DTA* uses offline training data, which can be laborious to gather. Furthermore, DTA* also makes the assumption of a search space with disjoint subtrees, searching independently under the competing actions that the agent might take next. Burns (2013) showed that it performs poorly compared to modern search algorithms like LSS-LRTA* that use a single lookahead search space.

More recently, a metareasoning approach has been employed for search in the form of regret minimization in MDPs (Tolpin and Shimony 2012). This work attempts to estimate the probabilistic gain of performing additional rollouts in a stochastic environment. Like DTA*, it considers the actions available at the agent's current state, and makes its decision based on the likelihood of additional computation showing that the currently best action is surpassed by a competitor. This is done by continually sampling the top level actions until the incumbent is deemed sufficiently more promising than its competitors.

A simplistic metareasoning analysis is used by the Ms. A* algorithm of Burns (2013) to determine whether to take identity actions and how much of the plan prefix to commit to. Like Dynamic $\hat{f}$, Ms. A* uses an unbiased inadmissible heuristic $\hat{h}$ and defines $\hat{f} = g + \hat{h}$. Define α as the action whose unbiased $\hat{f}$ is lowest, and β as a competing action to be taken. As indicated in Figure 3, for each action, Ms. A* interprets $\hat{f}$ as the expected value of a belief distribution regarding the location of the true f^* cost of the action and interprets the admissible f value as the truncated left edge of that distribution. (This is reminiscent of work on Bayesian reinforcement learning (Dearden, Friedman, and Russell 1998).) By definition, $\hat{f}(\alpha) \leq \hat{f}(\beta)$. If $f(\alpha) \leq f(\beta)$ as well, then Ms. A* concludes that either α is significantly better than β (left panel in the figure) or the two are so close that further search is not worth the negligible gain (middle panel). However, if $f(\beta) < f(\alpha)$ (right panel), then Ms. A* concludes that further search is warranted. While this analysis is quick and easy, it completely ignores the cost and likely effects of further search. The simplicity of Ms. A* can lead it to take identity actions when they do not result in improved results. O'Ceallaigh (2014) showed empirically that Ms. A* can sometimes outperform previous real-time algorithms, but that it is itself outperformed by the more principled algorithms presented below.

Deciding When to Act

We now turn to the first of the two metareasoning schemes we propose in this paper. It concerns 'identity actions', which are special actions available in some domains that the agent may take but that don't change the problem state, but merely allow the agent to delay acting. They allow the agent to continue searching from the same initial state as in the previous search iteration, without discarding the lookahead search space, and therefore they allow the agent to look further ahead and gather more information about the potential long-term effects of choosing the different actions applicable in the current state.

We call our algorithm $\hat{f}_{IMR}$, as it decides whether to take an identity action by using a metareasoning process like that of DTA*. It is a variant of Dynamic $\hat{f}$. Whenever an identity action is applicable in the current state, the algorithm attempts to estimate whether the time that could be spent planning during the identity action will be more than offset by the expected improvement in GAT resulting from being more likely to select a better action. If so, the learning step is skipped, an identity action is issued, and the search continues from where it left off.

More formally, if $t_{identity}$ is the duration of the identity action and B is the expected benefit of search (in terms of GAT reduction), then $\hat{f}_{IMR}$ will take the identity action iff

$$B > t_{identity}. \tag{1}$$

The value of $t_{identity}$ is known (likely selected by the system designer); it is B that is more difficult to estimate. Note that search will only provide benefit if, instead of taking the currently most promising action α, we select some other action β instead. So we must estimate the probability that, after searching, our estimate of β's expected cost has fallen below α's, and if so, by how much. Note that we choose actions based on their $\hat{f}$ values, so what we need to know is where α and β's $\hat{f}$ values are likely to be after we have performed more search. As in any metareasoning approach, we will need to make some significant assumptions and approximations in order to make our method practical.

We approach this problem from a perspective similar to that of Ms. A*: we view our belief about the value of an action a as a probability distribution over possible values, with $p_a(x)$ representing the probability density that a's true f^* value is x. Thinking back to Dynamic $\hat{f}$, we can see that further lookahead will decrease the error in a's backed up f value, represented in our $\hat{f}$ estimate as the d value that a inherits from its best frontier descendant b. If we view our $p_a(x)$ belief distribution, as we did with Ms. A*, as centered on $\hat{f}$ with variance controlled by $\bar{\epsilon} \cdot d(b)$, then we see that additional search will decrease our uncertainty about a's value, as we would expect. More specifically, because $\bar{\epsilon} \cdot d(b)$ represents our estimate of the expected error in a's f value ($\hat{f}(a) - f(a)$), we interpret it as an estimate of the standard deviation of p_a, and its square as the variance of the belief distribution: [1]

$$\sigma^2_{p_a} = (\bar{\epsilon} \cdot d(b))^2. \tag{2}$$

[1] An implementation detail: although the lookahead search of

Now what we need in order to compute B is an estimate of where $\hat{f}(a)$ might be located if we were to have performed additional search. We represent this too as a probability distribution, with $p'_{\bar{a}}(x)$ representing the probability density that $\hat{f}(a) = x$ after further search. We model $p'_{\bar{a}}$ as a Gaussian distribution. We first note that, because the current $\hat{f}(a)$ value is our best guess about the true value of a, we can use it as the mean of $p'_{\bar{a}}$. Second, we note that, if no further search were done, the variance of $p'_{\bar{a}}$ should equal that of p_a, and if we searched all the way to a goal, then the variance of $p'_{\bar{a}}$ would be zero, since we would know its true value. Therefore, we make the (admittedly strong) assumption that the variance of $p'_{\bar{a}}$ is

$$\sigma^2_{p'_{\bar{a}}} = \sigma^2_{p_a} \cdot (1 - \min(1, \frac{d_s}{d(b)})) \quad (3)$$

where d_s is the distance, in search steps, along the path to a goal that we expect to cover during the search. In other words, we take the variance of p_a as the variance of $p'_{\bar{a}}$, but scaled according to how far toward the goal we expect to get. To estimate d_s, we use the concept of expansion delay introduced by Dionne, Thayer, and Ruml (2011). Expansion delay estimates the average progress of a search along any single path. It is easily calculated as the average number of expansions from when a node is generated until it is expanded. Our implementation tracks path-based averages and uses the average computed during one iteration as the value for the next. Given the number of expansions that the search will perform within the duration of the candidate identity action,

$$d_s = \frac{expansions}{delay}. \quad (4)$$

This gives us all the ingredients for our belief distribution:

$$p'_{\bar{a}} \sim N(\hat{f}(a), \sigma^2_{p'_{\bar{a}}}). \quad (5)$$

Now that we can estimate how our beliefs about the values of actions might change with search, we can return to our central concern: estimating the possible benefit of search. If the value of the most promising action α were to become x_α and the value of some competing action β were to become x_β, the benefit would be

$$b(x_\alpha, x_\beta) = \begin{cases} 0 & \text{if } x_\alpha \leq x_\beta \\ x_\alpha - x_\beta & \text{otherwise} \end{cases} \quad (6)$$

because we would have done α if we had not searched. Now we just compute the expected value over our estimates of $p'_{\bar{\alpha}}$ and $p'_{\bar{\beta}}$:

$$B = \int_{x_\alpha} p'_{\bar{\alpha}}(x_\alpha) \int_{x_\beta} p'_{\bar{\beta}}(x_\beta) b(x_\alpha, x_\beta) dx_\beta dx_\alpha. \quad (7)$$

In the implementation tested below, we use a straightforward numerical integration with 100 steps.

$\hat{f}_{IMR}$ uses the same 'single-step global average' method for estimating $\bar{\epsilon}$ as Dynamic $\hat{f}$ did, preliminary studies (O'Ceallaigh 2014) showed that the alternative 'path-based correction' approach of Thayer, Dionne, and Ruml (2011) gave a better estimate when used to compute the variance of the desired belief distribution, so that method is used when computing variance.

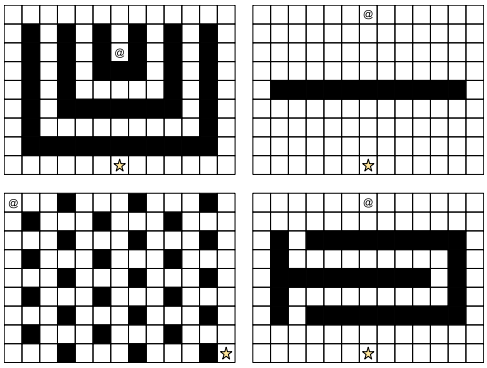

Figure 4: Handcrafted pathfinding problems. Clockwise from top left: nested cups, wall, slalom, uniform.

Experimental Results

To assess whether $\hat{f}_{IMR}$ behaves as we would expect, we constructed four simple handcrafted instances of grid pathfinding, where it is easy to assess algorithm behavior. To gauge whether $\hat{f}_{IMR}$ might be useful in practice, we then tested it on four larger more realistic real-time search benchmark domains.

Handcrafted Instances

Small versions of the four handcrafted pathfinding problems are sketched in Figure 4, with @ marking the start and a star marking the goal. Movement was four-way and the heuristic was Manhattan distance, ignoring obstacles. To see a full spectrum of behaviors and have a good basis for assessment, we tested A*, an off-line optimal search; RTA*, a simple real-time search; LSS-LRTA*, a modern real-time search; and Dynamic $\hat{f}$, the algorithm that $\hat{f}_{IMR}$ extends. We also tested a greedy hill-climbing algorithm that performs only one expansion and moves to the best child (equivalent to RTA* with a lookahead of one). In addition to measuring our central figure of merit, GAT, we also counted the number of short trajectories, when an algorithm chooses not to commit to all the actions along the best path to the lookahead frontier, and the number of identity actions executed. The two counts are identical for $\hat{f}_{IMR}$ but will differ for later algorithms in this paper. For all algorithms, the first search iteration is considered an identity action. For A*, all iterations are identity actions until the search is complete and the agent starts moving. To simplify reasoning about the algorithms, we disabled dynamic lookahead, so Dynamic $\hat{f}$ and $\hat{f}_{IMR}$ act like LSS-LRTA* and wait to search until the last committed action has begun executing. The lookahead (or equivalently, the time per action) was set to 10 expansions.

The first instance, nested cups, has sets of walls forming nested local minima which temporarily ensnare real-time searches. The full instance was 51×29 ($w \times h$) with corridors two spaces wide and a 3×3 space in the innermost cup. One would expect A* to outperform hill-climbing, for example, because A* expands each state at most once, while hill-climbing and similar real-time searches must revisit the same states repeatedly as they build up enough learning to

instance	algorithm	GAT	short	identity
cups	A*	166	90	90
	hill-climbing	3108	1	1
	RTA*	1666	1	1
	LSS-LRTA*	3500	1	1
	$\hat{f}$	5322	1	1
	$\hat{f}_{IMR}$	970	255	255
	$\hat{f}_{PMR}$	4238	646	1
	Mo'RTS	241	83	82
wall	A*	102	43	43
	hill-climbing	241	1	1
	RTA*	723	1	1
	LSS-LRTA*	523	1	1
	$\hat{f}$	717	1	1
	$\hat{f}_{IMR}$	101	31	31
	$\hat{f}_{PMR}$	441	100	1
	Mo'RTS	140	64	62
uniform	A*	29578	27180	27180
	hill-climbing	2999	1	1
	RTA*	2997	1	1
	LSS-LRTA*	3195	1	1
	$\hat{f}$	2997	1	1
	$\hat{f}_{IMR}$	2997	1	1
	$\hat{f}_{PMR}$	2997	1	1
	Mo'RTS	2997	1	1
slalom	A*	177	27	27
	hill-climbing	1974	1	1
	RTA*	2168	1	1
	LSS-LRTA*	382	1	1
	$\hat{f}$	638	1	1
	$\hat{f}_{IMR}$	161	6	6
	$\hat{f}_{PMR}$	4794	662	1
	Mo'RTS	161	6	6

Table 1: Performance on handcrafted pathfinding instances.

escape the minima. Experimental results are shown in Table 1, and indeed A* has the best performance, while most of the real-time methods suffer (the $\hat{f}_{PMR}$ and Mo'RTS algorithms will be introduced below). Notably, however, $\hat{f}_{IMR}$ is able to detect that the heuristic is deceptive and that planning ahead is useful, as it executes a substantial number of identity actions and performs 5.5 times better than the $\hat{f}$ method it is based on. We had expected $\hat{f}$ to perform well, but as it learns that the heuristic significantly underestimates, it increases $\bar{\epsilon}$, which causes it to behave more greedily. With an unreliable heuristic, $\hat{f}_{IMR}$ chooses more deliberation, which appears to be a better strategy.

The wall instance, where a single flat obstacle in the middle of the map blocks the goal, elicits similar behavior because the wall creates a single large local minimum. The full instance was 41×21 with a gap of one on either side of the wall. $\hat{f}_{IMR}$ executes many identity action and performs just as well as A*, while $\hat{f}$ is 7 times worse.

In the uniform instance, where small obstacles are uniformly distributed across the map, the local minima each require only one step to escape. The full instance was 1200×1200. We would expect the real-time algorithms to perform well, while A* would labor to determine the true optimal path among many close contenders. The results confirm these expectations, with $\hat{f}$ reaching the goal in a tenth of the time of A*. $\hat{f}_{IMR}$ recognizes that identity actions are not needed and matches the performance of the other real-time searches.

The final handcrafted instance, slalom, features a long and winding path down the center of the map to the goal, with a quicker option being to bypass the slalom via either side of the map. The full instance was 37×124 with the corridor 2 spaces wide and a 3-space margin around the outside. As the results indicate, the simple real-time algorithms commit to following the winding path while A* is able to find the outside path and reach the goal much faster. LSS-LRTA* and $\hat{f}$ start down the path but eventually turn back, while $\hat{f}_{IMR}$ quickly recognizes the deceptive heuristic, executes a few key identity actions, and reaches the goal even faster than A*.

To summarize, $\hat{f}_{IMR}$ appears able to successfully adapt to behave more like A* or more like hill-climbing as the situation requires. While these small benchmarks are very promising, it remains to be seen if the algorithm can perform well on full-scale benchmarks.

Larger Benchmarks

We used four benchmarks: grid pathfinding in the orz100d map from Dragon Age: Origins using the 25 start/goal combinations for which the optimal solution cost was highest (Sturtevant 2012), Korf's 100 instances of the 15-puzzle (Korf 1985), 100 randomly selected instances of a platformer-style video game (Burns, Kiesel, and Ruml 2013), and 25 randomly selected instances of a Frogger-style traffic avoidance game (O'Ceallaigh 2014). The video games are examples of domains where users expect agents to begin acting promptly and achieve goals quickly, while the sliding tile puzzle is a classic benchmark. While these domains are deterministic and fully observable, the time pressure of optimizing GAT highlights the trade-off between deliberation and acting. All but the traffic domain feature identity actions.

We used full dynamic lookahead for Dynamic $\hat{f}$ and $\hat{f}_{IMR}$. We tested at five different 'search speeds', varying the number of expansions allowed per unit of action duration in powers of 10 from 10^2 to 10^6 inclusive. For each instance, algorithms were given a limit of 7 GB of memory and 10 minutes of CPU time. No data point is plotted for any setting where an algorithm failed to reach the goal on one or more instances within the resource constraints.

We also tested RTA* and LRTA* (Korf 1990), DTA*, and Ms. A*, but their results were inferior and uninteresting, so we do not include them in the plots.

The left column of Figure 5 presents the results of $\hat{f}_{IMR}$, with each row showing a different domain. The x axis of each plot varies the search speed and the y axis shows the GAT, normalized as a factor of the GAT of an oracle that

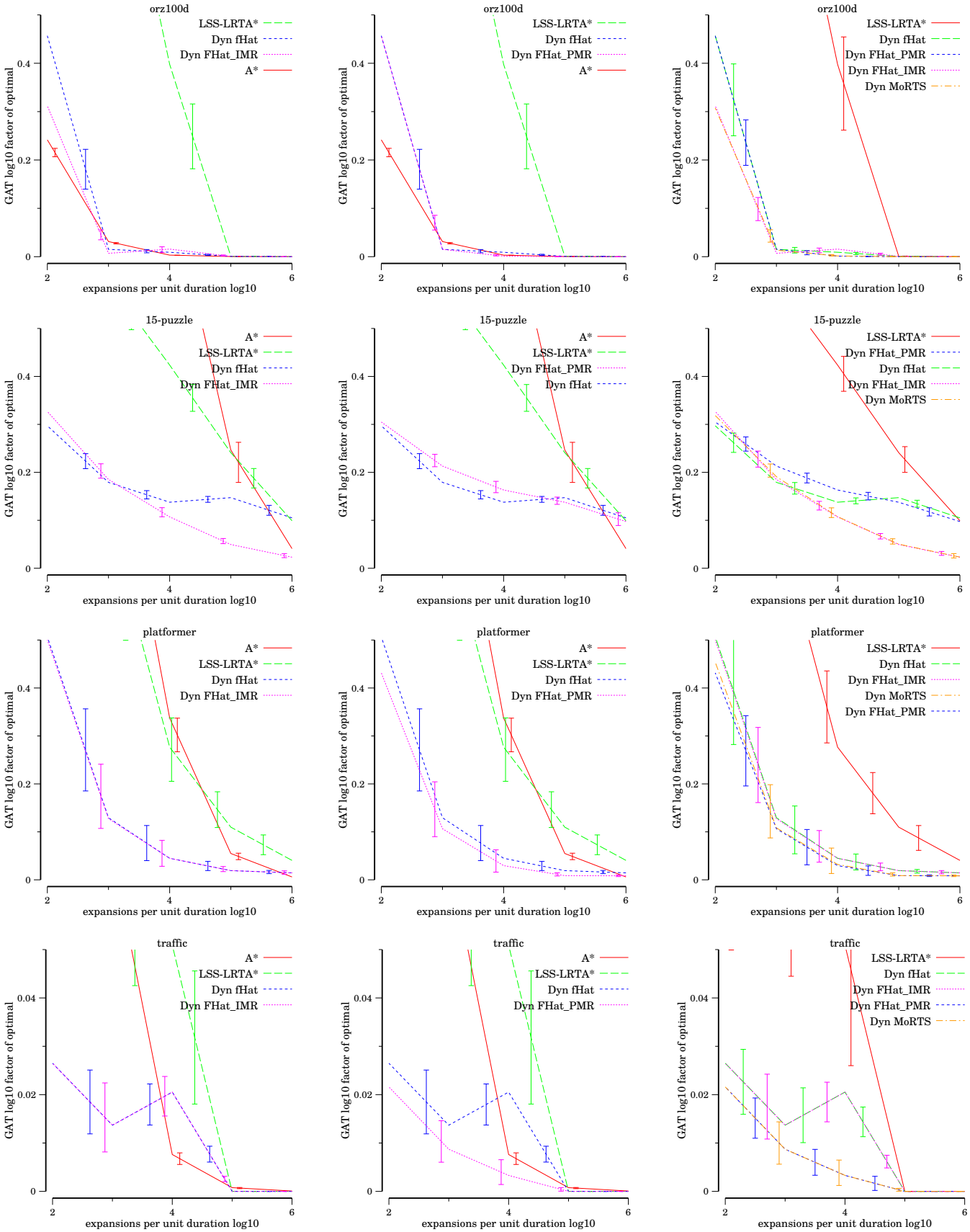

Figure 5: Goal achievement time as a function of search speed. One problem domain per row, different algorithms per column.

immediately commits to an optimal plan without searching, shown on a $\log_{10}$ scale. The normalization reduces the variance within each domain, as some instances are easier than others. Error bars show 95% confidence intervals on the mean over all the instances in the domain.

The top left panel shows the pathfinding domain. The heuristic is quite accurate and A* is hard to beat. But among the real-time algorithms, $\hat{f}_{IMR}$ shows a slight advantage when few expansions can be performed. The second panel shows the 15-puzzle, and $\hat{f}_{IMR}$ gives a pronounced advantage. The trade-off between behaving like A* when search is fast and behaving like Dynamic $\hat{f}$ when search is slow is evident. In the platformer domain, Dynamic $\hat{f}$ and $\hat{f}_{IMR}$ perform similarly, both better than LSS-LRTA*. And in the traffic domain, there are no identity actions, so $\hat{f}_{IMR}$ is identical to Dynamic $\hat{f}$, and both are better than LSS-LRTA*.

In summary, results in both the small handcrafted instances and the larger benchmarks suggest that a metareasoning approach to deciding when to commit to actions and when to plan further is quite promising, matching or outperforming existing real-time algorithms.

Deciding How Many Actions to Commit To

We now turn to the second of the two metareasoning schemes we propose in this paper. Recall that modern real-time search algorithms like LSS-LRTA* and Dynamic $\hat{f}$ explore a local search space and then commit to a plan prefix leading to a frontier node (line 5 in Figure 1). While $\hat{f}_{IMR}$ addressed the issue of whether to commit or search further, the issue we consider now is, given that we commit to acting, how much of the plan prefix should we commit to? If there exists some state s along the prefix P where the action α_s selected at s is not certainly better than an alternative β_s, it might prove worthwhile to cut P short such that it ends at s. This path prefixing operation allows the search to pay attention to promising paths which might otherwise be ignored. We call this algorithm $\hat{f}_{PMR}$, as it considers prefixes using metareasoning.

We use the same assessment of the benefit of search as in $\hat{f}_{IMR}$, computing B at the nodes along P and stopping at the first one for which search appears worthwhile. In this situation, we are not comparing the benefit B against the duration of an identity action, but rather against the more nebulous costs of 'starting search at a point before the frontier'. We tested two approaches to assessing these costs. The first was to assume that they were zero, leading us to commit to search at the first node for which B was positive. The second, and more successful, was to interpret the cost of stopping short of the frontier as the expected time required to regenerate the nodes from the new start state to the current lookahead frontier. We estimate this as the number of steps in the pruned suffix of P times the expansion delay, divided by the number of expansions per action duration. While this does estimate the amount of repeated work, it is not fully satisfactory, as this repeated work will likely be done concurrently with search and might not lead to increased GAT. However, it will certainly prevent the search from looking as far ahead in the search space as it would if we committed to the entire prefix, because a smaller prefix will have a smaller duration (line 6 in Figure 1).

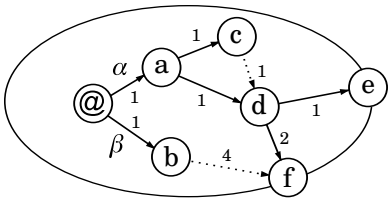

Figure 6: Useful (d) versus not useful (@, a) decision points.

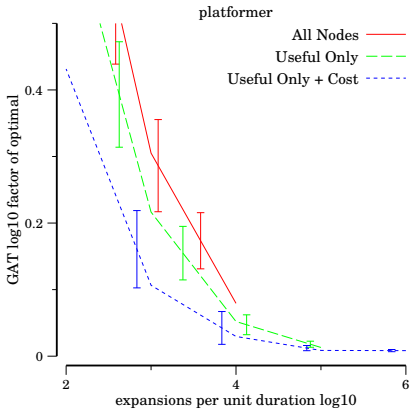

Figure 7: Variants of $\hat{f}_{PMR}$.

A second complication is that we only wish to consider performing more search at nodes along P that have successors that lie on best paths to different frontier nodes. While there will likely be alternative actions available at every step of the path, some of them may represent unnecessarily expensive temporary deviations from P, rather than truly useful alternatives. Figure 6 gives a concrete example. Edges drawn with solid lines also represent parent pointers back from the successor. The best path P is $\langle$@, a, d, e$\rangle$. Node d represents a useful decision point because it has multiple successors that lie along best paths to different frontier nodes. Node a is not a useful decision point, because c merely represents a longer way to reach the same frontier node as d. The agent's current state is also not a useful decision point, as b does not lie along the best known path to f, so there is no use in going that way. Computing usefulness just requires bookkeeping during the learning step, recording for each node the frontier node from which it inherits its value, or nil if the inheritance happened through a non-parent pointer (indicating that the node does not lie on a best path to the frontier). $\hat{f}_{PMR}$ then only performs its metareasoning at useful decision nodes along P, where at least two successors lie on best paths to distinct non-nil frontier nodes.

Experimental Results

Figure 7 illustrates how considering only nodes representing useful choices and considering even a rough cost for pruning a plan prefix improves the performance of $\hat{f}_{PMR}$.

The performance of $\hat{f}_{PMR}$ on the handcrafted instances is included in Table 1. While it led to some improvements over $\hat{f}$ on cups and wall, it performed poorly on slalom. Overall, it appears to provide less of a benefit than $\hat{f}_{IMR}$. Its behavior on the larger benchmarks is shown in the middle column of Figure 5. It performs comparably to Dynamic $\hat{f}$ except on traffic, where it outperforms all other algorithms. The performance on larger benchmarks is reminiscent of $\hat{f}_{IMR}$, in that $\hat{f}_{PMR}$ performs similarly to Dynamic $\hat{f}$ except on one domain, where it shows a pronounced advantage. This immediately suggests combining the two methods.

Mo'RTS

We investigated the combination of both the identity and prefix techniques in the same algorithm, which we call Mo'RTS (for metareasoning online real-time search, pronounced 'Moe RTS'). Mo'RTS checks if an identity action is applicable at the current state even if it is not a true decision node. Only if acting seems preferable to search is the best path checked for the prefix length to which the algorithm should commit.

The performance of Mo'RTS on the handcrafted instances is included in Table 1. Surprisingly, it outperforms both $\hat{f}_{IMR}$ and $\hat{f}_{PMR}$ on the cups instance, coming very close to A*'s performance. On the wall instance, it performs almost as well as $\hat{f}_{IMR}$, and the same as $\hat{f}_{IMR}$ on uniform and slalom. Performance on larger benchmarks is shown in the right column of Figure 5. On pathfinding and the 15-puzzle, where $\hat{f}_{IMR}$ is strong, Mo'RTS does just as well. And on traffic and platformer, where $\hat{f}_{PMR}$ is strong, Mo'RTS matches its performance. Overall, the results show significant improvement over the state-of-the-art LSS-LRTA* and Dynamic $\hat{f}$ algorithms.

Discussion

While the methods we introduce appear to work well across a variety of domains, they are based on several assumptions. First, both Dynamic $\hat{f}$ and metareasoning methods assume access to an inadmissible $\hat{h}$, which in this work we create using on-line debiasing. There is no strong practical theory that we currently know of to explain when such a method will result in a reasonable heuristic or not. Thayer (2012, Figure 4-3) and O'Ceallaigh (2014, Table 3.2) suggest that debiasing yields an inaccurate heuristic, yet it is clearly effective. Second, we make several assumptions in order to approximate $p'_{\bar{a}}$, including a Gaussian form, a linear variance reduction with lookahead, and a prediction of future expansion delay. In our limited investigations (O'Ceallaigh 2014), our Gaussian approximation of $p'_{\bar{a}}$ seems remarkably poor, and it is surprising that the metareasoning algorithms work as well as they do. There is likely much room remaining for improvements.

As we mentioned above, we do not currently have a fully satisfactory way in $\hat{f}_{PMR}$ to understand the implicit cost of choosing to commit to a plan prefix that stops short of the lookahead frontier. Using a short prefix results in less search time for the following iteration, which limits the number of actions to which that iteration may commit. Both $\hat{f}_{IMR}$ and $\hat{f}_{PMR}$ are essentially myopic. Explicitly reasoning about all this may be too expensive for on-line metareasoning.

In this study, we limited lookahead in node generations, not wall time. While this simplifies replication of results, it ignores the many issues necessary for deploying real-time search, such as predictable OS interrupt servicing and memory management. Considering an identity action is only done once per search iteration, if one is applicable at the agent's current state, and so the CPU overhead is likely negligible. Consider possible prefixes is done at potentially every node of the best path to the frontier, if all nodes are useful. (Computing usefulness adds negligible overhead, as explained above.) It remains to be seen whether this overhead is significant in practice, although in most domains there are many times more nodes generated than there would be along any single path to the frontier.

If the overhead of metareasoning could be made low enough, it may be beneficial to check more frequently whether the current lookahead gives sufficient confidence for committing to one or more actions. This would decouple the search iterations from the action start/end times. It would also provide an alternative approach to considering path prefix decisions.

Metareasoning has previously been used for directly guiding expansion decisions in off-line search, in which all planning occurs before any acting (Burns, Ruml, and Do 2013), and in contract search, where the planner faces a deadline (Dionne, Thayer, and Ruml 2011). It has also been used to decide which of multiple available heuristics to use in A* (Tolpin et al. 2013), IDA* (Tolpin et al. 2014), and CSP solving (Tolpin and Shimony 2011). This recent generation of work is fulfilling the early promise heralded by pioneers from the late 1980s such as Dean and Boddy (1988) and Russell and Wefald (1991).

Conclusion

In this work, we considered how metareasoning can be applied to the problem of concurrent planning and acting. We presented two methods for deciding how to commit to actions during a real-time search and investigated their behavior on both small easily-understood benchmarks and larger more realistic problems. Our techniques allow a single algorithm to dynamically adapt its behavior to the problem at hand, quickly committing to actions like greedy hill-climbing when possible or deliberating before acting like A* when necessary. Empirically, our methods match or outperform the state-of-the-art Dynamic $\hat{f}$ real-time search algorithm. We view this as an important success for metareasoning in combinatorial search, and we hope this work spurs more research in this area.

Acknowledgments

We gratefully acknowledge support from NSF (award 1150068), preliminary work by Sofia Lemons, code by Scott Kiesel and Ethan Burns, and discussions with Solomon Shimony, David Tolpin, and Ariel Felner.

References

Burns, E.; Kiesel, S.; and 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)*.

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *Journal of Artificial Intelligence Research* 47:697–740.

Burns, E. 2013. *Planning Under Time Pressure*. Ph.D. Dissertation, University of New Hampshire.

Dean, T. L., and Boddy, M. S. 1988. An analysis of time-dependent planning. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, 49–54.

Dearden, R.; Friedman, N.; and Russell, S. 1998. Bayesian Q-learning. In *Proceedings of AAAI-98*, 761–768.

Dionne, A. J.; Thayer, J. T.; and Ruml, W. 2011. Deadline-aware search using on-line measures of behavior. In *Proceedings of the Symposium on Combinatorial Search (SoCS-11)*. AAAI Press.

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.

Hernández, C.; Baier, J.; Uras, T.; and Koenig, S. 2012. Time-bounded adaptive A*. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-12)*, 997–1006.

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

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.

O'Ceallaigh, D. 2014. Metareasoning in real-time heuristic search. Master's thesis, University of New Hampshire.

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

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

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)*.

Thayer, J. 2012. *Heuristic Search Under Time and Quality Bounds*. Ph.D. Dissertation, University of New Hampshire.

Tolpin, D., and Shimony, S. E. 2011. Rational deployment of CSP heuristics. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 680–686.

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

Tolpin, D.; Beja, T.; Shimony, S. E.; Felner, A.; and Karpas, E. 2013. Toward rational deployment of multiple heuristics in A*. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI-13)*.

Tolpin, D.; Betzalel, O.; Felner, A.; and Shimony, S. E. 2014. Rational deployment of multiple heuristics in IDA. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI-14)*, 1107–1108.