Efficient Search with an Ensemble of Heuristics

Mike Phillips
Carnegie Mellon University

Venkatraman Narayanan
Carnegie Mellon University

Sandip Aine
Indraprastha Institute of Information Technology Delhi

Maxim Likhachev
Carnegie Mellon University

Abstract

Recently, a number of papers have shown that for many domains, using multiple heuristics in independent searches performs better than combining them into a single heuristic. Furthermore, using a large number of "weak" heuristics could potentially eliminate the need for the careful design of a few. The standard approach to distribute computation in these multi-heuristic searches is to rotate through the heuristics in a round-robin fashion. However, this strategy can be inefficient especially in the case when only a few of the heuristics are leading to progress. In this paper, we present two principled methods to adaptively distribute computation time among the different searches of the Multi-Heuristic A* algorithm. The first method, Meta-A*, constructs and searches a meta-graph, which represents the problem of finding the best heuristic as the problem of minimizing the total number of expansions. The second treats the scheduling of searches with different heuristics as a multi-armed bandit problem. It applies Dynamic Thompson Sampling (DTS) to keep track of what searches are making progress the most and continuously re-computes the schedule of searches based on this information. We provide a theoretical analysis and compare our new strategies with the round-robin method on a 12-DOF full-body motion planning problem and on sliding tile puzzle problems. In these experiments, we used up to 20 heuristics and observed a several times speedup without loss in solution quality.

1 Introduction

The efficiency of heuristic search methods, like A*, depends on an informative heuristic function. However, it is often difficult to design a single heuristic that captures all the complexities of a planning domain. It is generally easier to create a set of heuristics that each capture a part of the problem. A challenge arises when combining these into a single heuristic though. A typical approach is to use a weighted summation or take the maximum. However, there are many cases where the individual heuristic components oppose each other and combining them in these ways is no longer constructive and efficiency is lost. Recently, it has been shown that keeping the heuristics separate, each with their own search queue can perform significantly better [Aine *et al.*, 2014; Röger and Helmert, 2010; Isto, 1996]. Search progress can even be shared between queues (by sharing generated states) and the resulting synergy can cause these methods to solve much more difficult problems.

While these methods have been shown to solve significantly more complex problems, computational effort is distributed by rotating through the different heuristics in a *round-robin* fashion, causing a linear slowdown in the number of heuristics used, especially when only a small fraction of the heuristics are currently useful during the search. On the other hand, scaling these methods to a much larger number of heuristics would enable the use of an ensemble of *weak* heuristics, reducing time spent on engineering.

In this work, we present two methods for choosing which heuristic to follow at each state expansion. By giving more expansions to the more promising heuristics (determined in an online fashion) we can scale multi-heuristic methods up to use many more heuristics than before without suffering from a linear slowdown. We show that this allows for solving more difficult problems in shorter times.

The first method, Meta-A* constructs a meta-graph that represents the problem of finding the best heuristic to follow as the problem of minimizing the total number of expansions (as opposed to the underlying searches which are minimizing a user-chosen cost function). We then run A* on this meta-graph and the nodes in it chosen for expansion correspond to an expansion in underlying heuristic's search.

The second method treats the different heuristics as a multi-armed bandit problem where searches are given reward when an expansion makes progress (lowers the best seen heuristic value so far) and no reward otherwise. We then use Dynamic Thompson Sampling, DTS, [Gupta *et al.*, 2011] in order to tradeoff exploration among the different heuristics and exploitation of the heuristics which are currently working best.

On the theoretical side, we show the round-robin strategy is analogous to running Dijkstra's algorithm on the meta-graph developed in our Meta-A* method and therefore, under certain conditions, Meta-A* will never perform worse than round-robin in terms of the number of expansions made.

In this paper we will show how to apply the two methods to Multi-Heuristic A* (MHA*). This is a recent multi-heuristic search algorithm that can use a set of heuristics which are arbitrarily inadmissible and inconsistent, and is able to provide bounds on the suboptimality of found solutions as long as one heuristic in the set is known to be consistent [Aine *et al.*, 2014].

We provide experimental evaluations on robot motion planning and sliding tile domains. Our robotics domain is a 12 degree of freedom full-body motion planning problem for the PR2 robot. The planner can navigate the base of the robot as well as the arm in order to have the robot navigate to and reach into cluttered areas. The problem is considered to be extremely challenging in the motion planning community. We show an average speedup of 4.5 times over the round-robin method (while using 20 heuristics) with no loss in path quality. Additionally, we provide experiments on sliding tile puzzles which are a benchmark domain within the search community. In our experiments, we provide results on puzzles up to 10×10 in size. We attribute these results to allowing MHA* to manage a large number of heuristics through the use of Meta-A* and DTS.

2 Background: MHA*

Notation : In the following, S denotes the finite set of states of the domain. $c(s, s')$ denotes the cost of the edge between s and s', if there is no such edge, then $c(s, s') = \infty$. $\text{SUCC}(s) := \{s' \in S | c(s, s') \neq \infty\}$, denotes the set of all successors of s. $c^*(s, s')$ denotes the cost of the optimal path from state s to s'. $g(s)$ denotes the current best path cost from s_{start} to s and $h(s)$ denotes the heuristic for s, which is an estimate of the best path cost from s to s_{goal}. A heuristic is called *admissible* if it always underestimates the best path cost to s_{goal} and *consistent* if it satisfies, $h(s_{goal}) = 0$ and $h(s) \leq h(s') + c(s, s')$, $\forall s, s'$ such that $s' \in \text{SUCC}(s)$ and $s \neq s_{goal}$. OPEN denotes a priority queue, and is typically implemented as a min-heap.

Searching for optimal solutions in large state-spaces often leads to a dramatic increase in the time and memory required. WA* [Pohl, 1970] is a variant of A* that is often used to solve such large problems. It uses a priority function $f'(s) = g(s) + w * h(s)$ ($w > 1$) to provide a greedy flavor to the search, which often results in faster termination [Bonet and Geffner, 2001; Zhou and Hansen, 2002; Likhachev *et al.*, 2004]. WA* guarantees that the suboptimality of the solution is bounded by w times the optimal cost [Pearl, 1984] if $h(s)$ ($\forall s \in S$) is admissible, and does not require re-expansions to guarantee the bound if $h(s)$ is consistent [Likhachev *et al.*, 2004].

While WA* speeds up search for many applications, it relies heavily on the accuracy of the heuristic function. If the heuristic is subject to local minima however, then WA*'s performance can degrade severely [Hernández and Baier, 2012; Wilt and Ruml, 2012] owing to its greedy nature. Multi-Heuristic A* (MHA*) [Aine *et al.*, 2014] is a recently developed search algorithm that builds on the observation that while designing a single heuristic that is admissible, consistent and has shallow local minima is challenging for complex domains, it is often possible to design a number of weak (and possibly inadmissible) heuristics. MHA* uses multiple such (possibly) inadmissible heuristics to guide the search around local minima, by exploiting the synergy provided by the weak heuristics, each of which may be useful in different parts of the search space.

Algorithm 1 MHA*

```
1:  procedure MHA*()
2:      INITIALIZESEARCHES()
3:      INITIALIZEMETAMETHOD()
4:      while g(s_goal) > w_a * OPEN_0.MINKEY() do
5:          i ← CHOOSEQUEUE()
6:          k_0 ← OPEN_0.MINKEY()
7:          k_i ← OPEN_i.MINKEY()
8:          if k_i ≤ w_a * k_0 then
9:              s ← OPEN_i.TOP()   // i^th WA* search
10:             EXPANDSTATE(i, s)
11:         else
12:             s ← OPEN_0.TOP()   // Anchor search
13:             EXPANDSTATE(0, s)
14:         UPDATEMETAMETHOD(i)
```

Formally, MHA* (shown at a high level in Alg. 1) takes in one consistent heuristic (h_0), a set of arbitrary inadmissible heuristics ($h_1..h_n$) and two weight factors w_a and w_h (both ≥ 1). It then runs multiple WA* searches (with weight w_h) with the inadmissible heuristics ($h_1..h_n$) in a round-robin fashion (using separate priority queues for each search), while using the consistent heuristic h_0 in a separate WA* search, called the *anchor* search to control the round-robin strategy. Specifically, MHA* allows for expanding only those states s in WA* searches with inadmissible heuristics whose priority is within w_a of the smallest priority in the open list of the anchor search (line 8). This way, MHA* can guarantee completeness and bounded suboptimality ($w_a * w_h$) on the solution found. There are two variants of MHA* [Aine *et al.*, 2014], namely, Independent Multi-Heuristic A* (IMHA*) which uses independent g and h values for each search (a state expansion in search i only updates the i^{th} OPEN list) and Shared Multi-Heuristic A* (SMHA*) which uses independent h values but shares the g value among all the searches (each state expansion updates all OPEN lists). By sharing g values, SMHA* can use a combination of partial paths found by different searches to overcome local minima, thereby making it more powerful than IMHA*.

3 Meta-Algorithms

This section describes our contributions for efficiently handling searches with multiple heuristics. While they are described in the context of MHA* (the lines in gray, in Alg. 1), they are equally applicable for any search method that uses multiple heuristics.

3.1 Round-Robin

As formulated originally, MHA* uses the round-robin strategy (Alg. 2) of cycling through heuristics. On each iteration the queue with the next index is selected to perform an expansion and the algorithm loops back to the first after the last

Algorithm 2 Round-Robin (Original MHA*)

```
1: procedure INITIALIZEMETAMETHOD()
2:     queue ← 0
3: procedure UPDATEMETAMETHOD(i)  // No updates
4: procedure CHOOSEQUEUE()
5:     queue ← (queue + 1 mod n)
6:     return queue + 1
```

Algorithm 3 Meta-A*

```
1: procedure INITIALIZEMETAMETHOD()
2:     for i ∈ {1, 2, ..., n} do
3:         G_m[i] ← 0
4:         H_m[i] ← h_i(s_start)/(Δh_i)_max
5:         F_m[i] ← H_m[i]
6: procedure UPDATEMETAMETHOD(i)
7:     G_m[i] ← G_m[i] + 1
8:     H_m[i] ← (min_{s∈OPEN_i} h_i(s))/(Δh_i)_max
9:     F_m[i] ← G_m[i] + w_m * H_m[i]
10: procedure CHOOSEQUEUE()
11:     return arg min_i F_m[i]
```

queue takes its turn.

3.2 Meta-A*

The downfall of the round-robin approach is that it will continue to give equal computation time to all searches independently of how much progress each one is making. The first strategy, Meta-A*, treats the problem of selecting the next queue to expand as the problem of identifying the search that requires the fewest expansions to terminate. This method will keep track of how many expansions each search has made and an estimate of how many more each search has to go. It then combines these in an A* like fashion to decide which search to expand from on each iteration.

Since the approach is based on A*, it is easiest to understand in the context of a graph search. We define a meta-graph $\mathcal{G}_m$ which will represent how long it will take each search to terminate in isolation (we will assume the IMHA* case for now, so searches do not share progress). The meta-graph contains n independent chains (one for each search queue), as shown in Fig. 1. The length of a chain i, denoted by e_i, represents the number of expansions search i would take to terminate at the goal, and therefore, is initially unknown. Specifically, the j^{th} node in chain i (O_i^j) represents the j^{th} expansion of search queue i (it also represents the entire $OPEN$ list from search i after its j^{th} expansion).

Meta-A* runs an A* search on this meta-graph. The set of start states is $\{O_1^0, O_2^0, \ldots, O_n^0\}$ meaning that each of the n searches have made 0 expansions so far. The set of goal states in the meta-graph is $\{O_1^{e_1}, O_2^{e_2}, \ldots, O_n^{e_n}\}$, meaning that Meta-A* is going to try to reach a state where one of the n searches has terminated.

All edges in the meta-graph (O_i^j, O_i^{j+1}) have cost 1 which corresponds to search i making a single expansion. Also note that all vertices in the meta-graph have exactly 1 outgoing edge (except for the terminal vertex in each chain), again representing that from O_i^j, search i can expand another state

Algorithm 4 DTS

```
1: procedure INITIALIZEMETAMETHOD()
2:     for i ∈ {1, 2, ..., n} do
3:         h_best[i] ← h_i(s_start);  α[i] ← 1;  β[i] ← 1
4: procedure UPDATEMETAMETHOD(i)
5:     h'_best ← min_{s∈OPEN_i} h_i(s)
6:     if h'_best < h_best[i] then
7:         h_best[i] ← h'_best
8:         α[i] ← α[i] + 1   // Get reward 1
9:     else
10:        β[i] ← β[i] + 1   // Get reward 0
11:    if α[i] + β[i] > C then
12:        α[i] ← C/(C+1) α[i];  β[i] ← C/(C+1) β[i]
13: procedure CHOOSEQUEUE()
14:    for i ∈ {1, 2, ..., n} do
15:        r[i] ~ Beta(α[i], β[i])
16:    return arg max_i r[i]
```

from its $OPEN$ list (except from the terminal state $O_i^{e_i}$ because no more states can be expanded from search i after it has terminated). Note that because of the structure of the meta-graph, when running Meta-A* (A* on the meta-graph) the "meta $OPEN$ list" will always have constant size of n. In other words, there is always one state from each underlying search.

In order to run Meta-A* we must define the usual A* quantities g-value, h-value (heuristic), and f-value (priority). To avoid confusion with these values being used in the underlying searches, we will instead use G_m, H_m, F_m (g, h, and f used by Meta-A* in the meta-graph). $G_m(O_i^j)$ is cost from a start state (in this case O_i^0) to the state O_i^j. This is exactly j since we know all edges have cost 1. As pointed out earlier, the "meta $OPEN$ list" will contain exactly 1 state from search i. Specifically, it will be the most recently generated one, the one with the highest j seen so far. Therefore, we can simplify by just using $G_m[i]$ to mean the number of expansions made by search i so far. Similarly $H_m[i]$ is a heuristic estimate for how many more expansions search i will have to make before terminating. Additionally, if the estimate is admissible (does not overestimate) we will be able to show that Meta-A* finds an optimal solution within the meta-graph (we will explain what that means shortly). Estimating the remaining expansions is difficult. Therefore, we estimate it conservatively by guessing the heuristic of the underlying search will drop maximally with each expansion until reaching 0 (at which point we conservatively guess the goal is found as soon as the heuristic reaches 0). This brings up two assumptions of Meta-A*. The first is that we assume that the heuristics of the underlying searches are 0 at the goal state. The other is that for each heuristic h_i there is a known bound on the maximum decrement $(\Delta h_i)_{max}$, in the heuristic along any single edge the search will encounter. Specifically,

$$(\Delta h_i)_{max} = \max_{s \in \mathcal{G}, s' \in succ(s)} (h_i(s) - h_i(s')) \qquad (1)$$

States with infinite h_i values are ignored in this definition, since they will never be expanded in the i^{th} search. We also assume that the heuristic is not trivial, i.e, h_i is not identi-

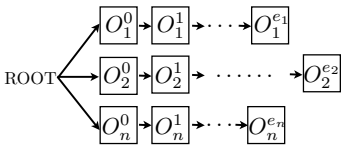

Figure 1: The Meta-Graph $\mathcal{G}_m$.

cally zero everywhere. This ensures $(\Delta h_i)_{max}$ is not zero. Note that if h_i is consistent, $(\Delta h_i)_{max}$ can be replaced by the largest edge cost in the graph, a typically known quantity.

We can use $(\Delta h_i)_{max}$ as a rescaling factor to change an estimate $h_i(s)$ (an estimate for the remaining cost to the goal) into the remaining number of edges from s to the goal (by dividing by $(\Delta h_i)_{max}$). Therefore, conservatively estimating the remaining expansions for search i ($H_m[i]$) will involve choosing the minimum heuristic value in the i^{th} search's $OPEN$ list and dividing it by $(\Delta h_i)_{max}$ as shown on line 8 of Algorithm 3.

On each iteration, Meta-A* will choose a search to perform an expansion. It chooses the search i with minimum priority defined as $F_m[i] = G_m[i] + w_m * H_m[i]$. For now, assume $w_m = 1$. In this case $G_m[i]$ (expansions made) is added to $H_m[i]$ (conservative estimate of remaining expansions) which gives priority $F_m[i]$ (a conservative estimate of expansions needed by the i^{th} search from its start to termination). In fact, for $w_m = 1$ (and when running IMHA*), throughout the execution of Meta-A*, $F_m[i]$ is always less or equal to the initially unknown but static e_i. Our theoretical analysis will show how this leads to Meta-A* never performing more expansions than Round Robin (under the conditions mentioned). Note that round-robin is a special case of Meta-A* where $H_m[i] = 0 \forall i$. On each expansion, round-robin chooses the search with the fewest expansions so far (minimal $G_m[i]$). Therefore, round-robin also searches the meta-graph, but by using a heuristic, Meta-A* performs better (like the relationship between Dijkstra's algorithm and A*).

Finally, much like how weighted A* inflates the heuristic term by a scalar larger than 1 in Meta-A* we have a scalar w_m which can be set larger than 1 in order to be more heuristic (H_m) driven. This tends to make Meta-A* much faster.

In Alg. 3 we can see that UPDATEMETAMETHOD (which is called after an expansion from search i) increments the G_m for that search, updates the meta heuristic $H_m[i]$, and then recomputes the priority F_m for search i. When CHOOSEQUEUE is called, we use the search with smallest priority.

Theoretical Analysis

For theoretical analysis, we assume that Meta-A* is used in conjunction with IMHA* (maintains an independent queue for each heuristic). For IMHA*, we additionally assume that the anchor inflation w_a is large enough that the anchor search never runs (no expansions from OPEN_0). All analysis applies for $w_m = 1$, and can be generalized for other values of w_m, as done for WA*. We assume that $h_i(s_{goal}) = 0, \forall i \in \{1, 2, \ldots, n\}$, although they could be arbitrarily inadmissible elsewhere.

Lemma 1 (Admissible Meta-Heuristic)**.** *The heuristic H_m as computed in Alg. 3 is an admissible heuristic on the meta-graph $\mathcal{G}_m$, irrespective of whether h_i is admissible or not.*

Proof. We have,

$$H_m(O_i^k) = \frac{\min_{s \in O_i^k} h_i(s)}{(\Delta h_i)_{max}} \tag{2}$$

We will show that $H_m(O_i^k)$ does not overestimate $H_m^*(O_i^k)$, the true cost-to-go for O_i^k, to prove that H_m is admissible on $\mathcal{G}_m$. Let $\Pi(s_1, s_2)$ denote the set of all paths between two states s_1 and s_2. Define

$$\begin{aligned} s' &= \arg\min_{s \in O_i^k} h_i(s) \qquad (3)\\ s'' &= \arg\min_{s \in O_i^k} \min_{\pi \in \Pi(s, s_{goal})} \text{NUMEDGES}(\pi) \\ \pi^* &= \arg\min_{\pi \in \Pi(s'', s_{goal})} \text{NUMEDGES}(\pi) \\ e'' &= \text{NUMEDGES}(\pi^*) \end{aligned}$$

In the above, s' is the state based on which we compute H_m and s'' is the state in OPEN_i which is the least number of edges away from the goal s_{goal}. Clearly, e'' does not overestimate the number of expansions needed for search i to terminate, continuing from O_i^k. That is,

$$e'' \leq H_m^*(O_i^k) \tag{4}$$

Assume for the sake of contradiction that $H_m(O_i^k) > e''$. Let $\pi^* = (s_0, s_1, s_2, \ldots, s_{e''})$, where $s_0 = s''$ and $s_{e''} = s_{goal}$. Since $h_i(s_{goal}) = 0$,

$$\begin{aligned} h_i(s'') &= h_i(s'') - h_i(s_{goal}) \\ h_i(s'') &= \sum_{j=0}^{e''-1} h_i(s_j) - h_i(s_{j+1}) \\ h_i(s'') &\leq e'' * (\Delta h_i)_{max} && \text{(from Eq. (1))} \\ h_i(s'')/(\Delta h_i)_{max} &\leq e'' \\ h_i(s'')/(\Delta h_i)_{max} &< H_m(O_i^k) && \text{(assumption)} \\ h_i(s'')/(\Delta h_i)_{max} &< h_i(s')/(\Delta h_i)_{max} && \text{(from Eq. (2))} \\ h_i(s'') &< h_i(s') \end{aligned}$$

This contradicts our definition of s' (Eq. (3)). Hence, the assumption $H_m(O_i^k) > e''$ is incorrect and we have

$$\begin{aligned} H_m(O_i^k) &\leq e'' \\ H_m(O_i^k) &\leq H_m^*(O_i^k) && \text{(using Eq. (4))} \end{aligned}$$

Therefore, the heuristic H_m does not overestimate the actual cost-to-go on $\mathcal{G}_m$. ■

Theorem 1 (Bounded Expansions)**.** *The total number of state expansions resulting from this strategy is bounded from above by $n \cdot e^*$, where $e^* = \min_i e_i$.*

Proof. Consider an A^* search running on a tree where the optimal solution cost is g^*. Let E be the number of expansions made by A* search before the goal is expanded. Define $\mathcal{S} = \{s | g(s) \leq g^*\}$. Then, for A* search with an admissible

heuristic on this tree, $E \leq |\mathcal{S}|$ (re-expansions are not possible because the graph is a tree). We know from Lemma 1 that Meta-A* uses an admissible heuristic, and that $\mathcal{G}_m$ is a tree. Further, for $\mathcal{G}_m$, $g^* = e^*$ and $|\mathcal{S}| = n \cdot e^*$. Thus, the expansions made by Meta-A* is bounded from above by $n \cdot e^*$. ■

Note, the upper bound on expansions is exactly the same as the best-case for the round-robin strategy. Thus, the proposed strategy is no worse than round-robin. However, the best case for Meta-A* is e^*, better than round-robin by a factor n.

Theorem 2 (Best Heuristic in Hindsight)**.** *When the algorithm terminates, the last search that updates $g(s_{goal})$ is the best heuristic in hindsight (h_{i^*}, where $e_{i^*} = e^*$). In other words, running search i^* alone would have been optimal in hindsight.*

Proof. The first part follows from A^*'s. Since the optimal solution cost for the meta-graph is e^*, goal state $O_{i^*}^{e^*}$ will be expanded ahead of all other possible goal states. Therefore, the goal state in the original graph s_{goal} is first expanded from OPEN$_{i^*}$, enabling us to determine i^*. ■

Theorem 3 (Optimal Efficiency)**.** *Any other algorithm or strategy for selecting P_i that uses the same heuristic as Meta-A* (H_m) must expand at least as many states in total as expanded by Meta-A*, to guarantee that it has found the best heuristic in hindsight.*

Proof. This directly follows from the optimal efficiency property of running A^* on $\mathcal{G}_m$. ■

3.3 Dynamic Thompson Sampling

In the second method, we treat the problem of selecting the search queue to expand as the problem of selecting the arm to pull in a multi-arm bandit (MAB) problem. Whether an expansion "makes progress" determines if the bandit gives reward or not. Specifically, the i-th queue, q_i will keep track of the best heuristic value seen so far, $h_{best}[i]$ (the "closest" that queue has been to the goal according to h_i). If an expansion generates a successor with a heuristic value less than $h_{best}[i]$, reward of 1 is given and otherwise reward of 0.

Our instance of the MAB problem is called Dynamic Bandits since the internal parameters (i.e. the likelihood of giving reward) changes over time. We chose to apply Dynamic Thompson Sampling (DTS), a recent approach which reacts quickly to changing bandits [Gupta *et al.*, 2011]. The original Thompson Sampling (TS) which DTS is based on, is one of the earlier algorithms for solving MAB [Thompson, 1933]. TS maintains a beta distribution (with shape parameters $\alpha[i]$ and $\beta[i]$) for each bandit over the likelihood of the internal Bernoulli parameter. For this case of static Bernoulli bandits, recent work has shown strong theoretical bounds on expected worst case performance [Agrawal and Goyal, 2012].

The application of DTS to our problem is shown in Algorithm 4. TS and DTS are almost the same, in fact, DTS only adds lines 11-12. In INITIALIZEMETAMETHODS, we initialize the α and β parameters for each queue's beta distribution to 1 (a uniform distribution). If a priori knowledge about heuristic performance was known, these priors could be modified appropriately. In CHOOSEQUEUE TS chooses a bandit to play by drawing a sample from each beta distribution ($r[i]$ drawn from $Beta(\alpha[i], \beta[i])$) and selecting the bandit with the highest sample. In UPDATEMETAMETHOD, the reward is determined and the beta distributions are updated. On lines 5-6 we see if the expansion from queue i resulted in an improvement in the queue's $h_{best}[i]$. If so, we update it and queue i gets reward 1 which corresponds to incrementing $\alpha[i]$ (lines 7-8), otherwise queue i gets no reward which corresponds to incrementing $\beta[i]$ (line 10). If the algorithm stopped here, this would be Thompson Sampling. However, what makes DTS are lines 11-12. The parameter $C \geq \alpha + \beta$ controls how much history we care about. If $\alpha + \beta$ exceeds C (it will exceed by exactly 1) we normalize so that they sum to C again.

DTS quickly learns which heuristics are leading to search progress and spends more time expanding from them. When the performance of one of the heuristics being exploited drops (i.e., it hits a local minima and stops making progress) its beta distribution quickly skews toward β (due to C limiting history) and DTS tries other heuristics. When all heuristics do poorly, all distributions look similar and DTS is essentially choosing queues uniformly at random.

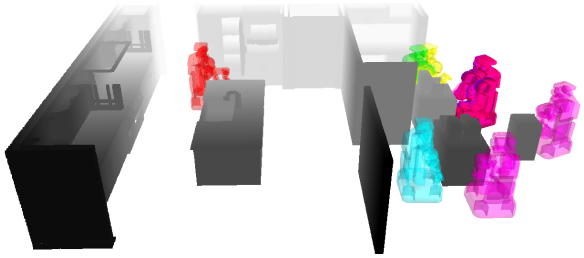

Figure 2: The kitchen domain for our experiments. Note the randomly placed tables on the right with randomized clutter on top. In the middle is a narrow door that separates the two rooms. The different colored robots show node expansions from our 20 OPEN lists prioritized by different heuristics.

4 Experimental Results

4.1 Motion Planning

DTS and Meta-A* are first evaluated in a 12 degree of freedom (DoF) full-body robot motion planning domain for the PR2 (a dual-arm mobile robot). The objective is for the planner to generate a collision free motion for the robot to approach and pick up objects on cluttered tables in a kitchen environment. Specifically, the planner controls the base position and orientation (x, y, θ), height of prismatic spine which raises and lowers the torso, 6DoF pose of the gripper in the robot's frame, and two arm "free angles" (direction of elbow).

Each state in the graph we plan on corresponds to a complete robot configuration. From any state the robot has a set of *motion primitives* it can apply, which are short kinematically feasible motions [Likhachev and Ferguson, 2008]. Collision free motion primitives connect pairs of states, thereby representing edges in our graph. While there is a single start state, the goal state is underspecified as any state that results in the gripper reaching the object meets the goal conditions.

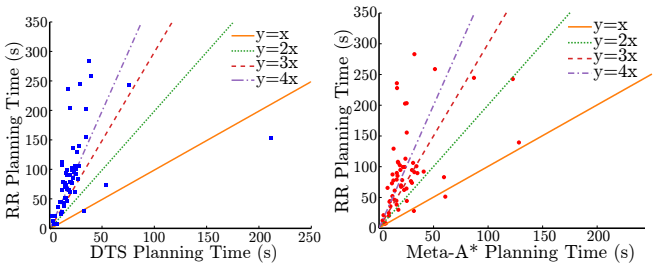

Figure 3: In the left (right) plot, each data point shows a DTS (Meta-A*) planning time against a round-robin planning time on the same trial. The four reference lines show where points fall for 1, 2, 3, and 4 times speedups.

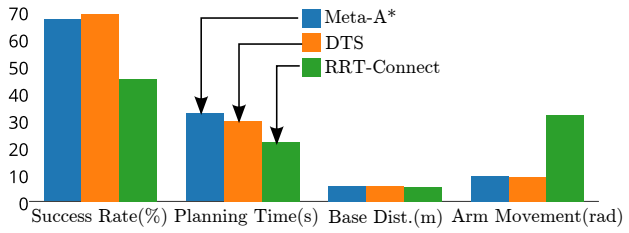

Figure 4: Comparison with RRT-Connect for full-body motion planning on the PR2 robot. Shown are the avg. planning times and avg. distances moved by the robot's base and arm.

The domain (Fig. 2) is challenging due to high dimensionality, cluttered tables, and narrow passages which must be crossed (the robot's base just fits through the doorway and only with arm tucked). A multi-heuristic search is ideal for dealing with many, often conflicting heuristics (e.g. wanting to extend the arm when reaching for a goal, while wanting to tuck when going through a door) In most cases, several heuristics are needed at different points during planning to find a solution quickly. The proposed methods get speedups by switching between these heuristics.

We designed 20 heuristics (19 + 1 anchor) to help the robot solve the problems efficiently. 16 of the heuristics guide the base's xy position while requiring different fixed base headings and a tucked arm. These heuristics help navigation in tight spaces, but can't reach for the goal. 3 heuristics then guide the arm to the goal with or without guiding the base to a specific pose within arm's reach of the goal. Note, almost all of these heuristics are highly inadmissible.

100 trials were generated with the following randomizations: positions of two kitchen tables and with random clutter every 10 trials, goal poses for the gripper to reach over tables, and the robot's start configurations.

We ran SMHA* with DTS ($C = 10$), Meta-A* ($w_m = 10$), and round-robin on all 100 trials. All three used $w_h = 25$ and $w_a = 4$. We also ran RRT-Connect, which is a popular (and considered among the fastest) sampling-based algorithm for motion planning [Kuffner and LaValle, 2000]. All methods were given 5 minutes to plan after which it is called a failure. A simple shortcutter was run on all paths after planning. This is crucial for RRT-Connect which otherwise does not minimize cost. For search methods like MHA* this is also useful to remove discretization artifacts from paths. 87 of the 100 trials were solved by at least 1 method.

Fig. 3 shows scatter plots of planning times. Each point shows the planning time of one of our methods (left plot is DTS and right plot is Meta-A*) against the planning time for round robin on the same query. A point above the y=x line means there is a speedup over round-robin. A point on the y=2x line means it is twice as fast, and so on. For DTS, most points concentrate around the 4 times speedup; for Meta-A*, 3 and 4 times. Both methods have an average speedup around 4.5. Since realtime (millisecond) planning is needed in robotics, speedups on simple problems are as important as on complex problems. All methods solve about 65 of 100 trials (round-robin solves slightly fewer), and solution costs vary less than 10% across methods. Therefore, substantial speedup is seen with small changes in solution quality.

Since DTS is a randomized method we ran all trials five times to compute variance. Success rate had a standard deviation of 1.41 trials. The average standard deviation of planning times across all trials was 10.7s. However, when we removed the trials that contained outliers (7 such trials), the average standard deviation was only 0.88s.

In Fig. 4, we can see how Meta-A* and DTS compare to RRT-Connect. Most notably, both of the new methods have a significantly higher success rate. RRT-Connect is faster than both methods on average (over the trials solved by both methods) but by less than a factor of 1.5. In terms of path quality, the robot's base travels roughly the same distance for all three methods. However, RRT-Connect generates arm motions that are over 3 times longer than DTS and Meta-A*. Overall, search methods have a higher success rate, somewhat slower, but have better solution quality.

4.2 Sliding Tile Puzzles

Subsequently, DTS and Meta-A* are applied to IMHA* and SMHA* with varying heuristic sets on large sliding tile puzzles (8×8, 9×9 and 10×10). Manhattan distance plus linear conflicts is used as consistent heuristic, h_0, and additional (inadmissible) heuristics are as follows. For a given puzzle size, we generate a database of 1000 different solved configurations by performing a random walk of k (a random number between 2 and 10 times the puzzles size) steps from s_{goal}. For each configuration, the path to goal and its cost, k, is stored. To generate n inadmissible heuristics, we cluster the database into n parts using the heuristic difference between two configurations as the distance metric. To solve a given instance with configuration sc, one target configuration, tc_i, is chosen per cluster such that the heuristic distance between sc and tc_i is minimum. Once tc_i is chosen, inadmissible heuristic h_i for any state s is computed by $h_i(s) = w_1 * h_0(s, tc_i) + cost(tc_i)$, where w_1 is the inflation factor used by MHA*.

We used 100 randomly generated instances of puzzles for each size as our benchmark suite. Table 1 shows the number of problems solved by MHA*s with different meta-level strategies for 4 heuristic set sizes: 5, 9, 13 and 17. For this domain, we ran DTS with $C = 1000$ and Meta-A* with $w_m = 100$. We also ran WA* (without re-expansions) on the same set of problems with h_0 as the heuristic and $w = 10$. WA* could only solve a small fraction of the problem in-

		Round-Robin				DTS				Meta-A*			
Algo.	Size	5	9	13	17	5	9	13	17	5	9	13	17
IMHA*	8×8	97	99	100	100	98	100	100	100	97	100	100	100
	9×9	72	82	89	95	79	94	98	99	74	84	98	99
	10×10	32	33	41	32	35	44	50	53	34	37	50	53
SMHA*	8×8	99	99	100	100	100	100	100	100	100	100	100	100
	9×9	79	89	94	90	84	98	100	100	83	94	99	100
	10×10	47	60	34	33	60	72	70	73	49	66	68	69

Table 1: Number of instances solved by MHA*s with round-robin, DTS and Meta-A*. Time Limit = 180s, bound = 10.

stances (66 for 8×8, 31 for 9×9 and 10 for 10×10).

In contrast, MHA* (with $w_a = 2$, $w_h = 5$, bound $= w_a \times w_h = 10$) performs better, showing the benefit of multiple heuristics. Relative to the round-robin results, MHA* solves more problems as the number of heuristics increases. Sometimes, performance degrades due to overhead (e.g. 10×10 puzzles with round-robin). SMHA* solves 47 instances with 5 heuristics (and 60 with 9), but overhead begins to dominate at 13 heuristics and only 34 instances are solved. Similarly, IMHA* (10×10 puzzles) solves 41 instances with 13 heuristics but only 32 instances with 17 heuristics. It may be noted that although in general SMHA* (with round-robin) outperforms IMHA* (with round-robin), we observe that for 13 heuristics (Table 1) the trend is reversed. This is mainly due to the fact that whenever a search visits a state in SMHA* for the first time, all the heuristics need to be computed, whereas IMHA* only computes one heuristic at a time. At times this extra computation can overshadow the benefits of sharing, thus causing degradation.

For each heuristic set size, DTS and Meta-A* solve more problems than round-robin, clearly highlighting the value of meta-level scheduling over round-robin. Also, in most cases, the number of problems solved increases monotonically with an increase in heuristic set size, indicating that DTS and Meta-A* can effectively control the overhead of using more heuristics. In terms of runtime, DTS and Meta-A* had similar speedups over round-robin with an average 1.8 times for SMHA* and 1.5 for IMHA*. In terms of solution costs, all the strategies perform comparably. Overall, the solution costs do not deviate by more than 5% and the average solution cost is practically the same for all methods (average solution costs obtained in our experiments are the following, 1367 for 8×8, 1828 for 9×9, and 2127 for 10×10 puzzles.

5 Related Work

The idea of a meta-level reasoner to improve planning is not new. Planning portfolios have been used when faced with a battery of planning problems. One particularly relevant example uses a bandit framework to choose which planner to run on each trial [Valenzano *et al.*, 2012]. The primary difference is that we are doing this within a single search.

With respect to applying bandit methods within a single search, the popular UCT algorithm has been used to select the next move in large MDPs or game trees [Kocsis and Szepesvri, 2006]. We differ in that we want to find entire solutions with bounds on solution cost, instead of a first move.

Exceptions to these standard applications use UCT to solve CSPs and boolean satisfiability problems [Loth *et al.*, 2013; Previti *et al.*, 2011]. These problems have different properties than ours (search trees in their problems have known, relatively shallow depth). We also differ in using bandits to choose heuristics instead of a node to explore next.

One different way of viewing planning as a bandit problem is that expanding the min priority state is exploiting the best known bandit [Valenzano *et al.*, 2014]. The authors then incorporate exploration by using an ε-greedy strategy which chooses a random state for expansion with probability ε.

In [Mellor and Shapiro, 2013], bandit parameters switch between 2 states (good and bad reward state). However, the method depends on a model of how likely a bandit is to stay or switch. In our domain it corresponds to the size of local minima, which vary wildly between problems.

EES [Thayer and Ruml, 2011] and QA(ϵ) [Chakrabarti *et al.*, 1989] attempt to achieve faster planning times by using an inadmissible estimate of the number of expansions, in addition to a consistent heuristic for bounds on solution quality. While Meta-A* uses a similar idea, it is different in that it applies for an ensemble of heuristics, and attempts to minimize the total expansions made by all the searches.

6 Acknowledgements

This work was supported by NSF grant IIS-1409549 and ONR DR-IRIS MURI grant N00014-09-1-1052.

7 Conclusions

In this work we have presented two methods to help multi-heuristic searches scale to large ensembles of heuristics. Both the Meta-A* and DTS methods show several times speedups over the naive round-robin rotation over heuristics. They do so with negligible change in solution quality and are both easy to implement. We also provide a strong theoretical analysis of Meta-A* for the IMHA* case.

While we used our methods in the context of MHA*, we believe they can be applied with equal success to other multi-heuristic searches [Röger and Helmert, 2010]. It is future work to confirm this experimentally. Our future work for Meta-A* is to see if any guarantees can be extended to SMHA*. For DTS we will look into proving expected worst case performance bounds.

Another future direction is parallelization of the two methods presented. Interesting questions arise when considering how often parallel search queues should share information in order to minimize synchronization overhead.

References

[Agrawal and Goyal, 2012] Shipra Agrawal and Navin Goyal. Further optimal regret bounds for thompson sampling. *CoRR*, abs/1209.3353, 2012.

[Aine *et al.*, 2014] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic A*. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[Bonet and Geffner, 2001] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[Chakrabarti *et al.*, 1989] P. P. Chakrabarti, Sujoy Ghose, A. Pandey, and S. C. De Sarkar. Increasing search efficiency using multiple heuristics. *Inf. Process. Lett.*, 30(1):33–36, 1989.

[Gupta *et al.*, 2011] N. Gupta, O.-C. Granmo, and A Agrawala. Thompson sampling for dynamic multi-armed bandits. In *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, volume 1, pages 484–489, Dec 2011.

[Hernández and Baier, 2012] C. Hernández and J. A. Baier. Avoiding and escaping depressions in real-time heuristic search. *J. Artif. Intell. Res. (JAIR)*, 43:523–570, 2012.

[Isto, 1996] Pekka Isto. Path planning by multiheuristic search via subgoals. In *Proceedings of the 27th International Symposium on Industrial Robots, CEU*, pages 71272–6, 1996.

[Kocsis and Szepesvri, 2006] Levente Kocsis and Csaba Szepesvri. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[Kuffner and LaValle, 2000] James J. Kuffner and Steven M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *ICRA*, pages 995–1001. IEEE, 2000.

[Likhachev and Ferguson, 2008] M. Likhachev and D. Ferguson. Planning long dynamically-feasible maneuvers for autonomous vehicles. In *Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, USA, June 2008.

[Likhachev *et al.*, 2004] M. Likhachev, G. J. Gordon, and S. Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[Loth *et al.*, 2013] Manuel Loth, Michèle Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based Search for Constraint Programming. In Christian Schulte, editor, *International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *LNCS*, pages 464–480, Uppsala, Suède, September 2013. Springer Verlag.

[Mellor and Shapiro, 2013] Joseph Mellor and Jonathan Shapiro. Thompson sampling in switching environments with bayesian online change point detection. *CoRR*, abs/1302.3721, 2013.

[Pearl, 1984] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

[Pohl, 1970] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[Previti *et al.*, 2011] Alessandro Previti, Raghuram Ramanujan, Marco Schaerf, and Bart Selman. Monte-carlo style uct search for boolean satisfiability. In *Proceedings of the 12th International Conference on Artificial Intelligence Around Man and Beyond*, AI*IA'11, pages 177–188, Berlin, Heidelberg, 2011. Springer-Verlag.

[Röger and Helmert, 2010] Gabriele Röger and Malte Helmert. The more, the merrier: Combining heuristic estimators for satisficing planning. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *ICAPS*, pages 246–249. AAAI, 2010.

[Thayer and Ruml, 2011] Jordan Tyler Thayer and Wheeler Ruml. Bounded suboptimal search: A direct approach using inadmissible estimates. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 674–679, 2011.

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

[Valenzano *et al.*, 2012] R. Valenzano, H. Nakhost, M. Müller, J. Schaeffer, and N. Sturtevant. Arvandherd: Parallel planning with a portfolio. *European Conference on Artificial Intelligence (ECAI 2012)*, 2012.

[Valenzano *et al.*, 2014] Richard Valenzano, Nathan Sturtevant, Jonathan Schaeffer, and Fan Xie. A comparison of Knowledge-Based GBFS enhancements and knowledge-free exploration (short paper). In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[Wilt and Ruml, 2012] C. M. Wilt and W. Ruml. When does weighted A* fail? In *SOCS*. AAAI Press, 2012.

[Zhou and Hansen, 2002] R. Zhou and E. A. Hansen. Multiple Sequence Alignment Using Anytime A*. In *Proceedings of 18th National Conference on Artificial Intelligence AAAI'2002*, pages 975–976, 2002.