# Dynamic Multi-Heuristic A*

Fahad Islam[†], Venkatraman Narayanan[†] and Maxim Likhachev[†]

*Abstract*— **Many motion planning problems in robotics are high dimensional planning problems. While sampling-based motion planning algorithms handle the high dimensionality very well, the solution qualities are often hard to control due to the inherent randomization. In addition, they suffer severely when the configuration space has several 'narrow passages'. Search-based planners on the other hand typically provide good solution qualities and are not affected by narrow passages. However, in the absence of a good heuristic or when there are deep local minima in the heuristic, they suffer from the curse of dimensionality. In this work, our primary contribution is a method for dynamically generating heuristics, in addition to the original heuristic(s) used, to guide the search out of local minima. With the ability to escape local minima easily, the effect of dimensionality becomes less pronounced. On the theoretical side, we provide guarantees on completeness and bounds on suboptimality of the solution found. We compare our proposed method with the recently published Multi-Heuristic A* search, and the popular RRT-Connect in a full-body mobile manipulation domain for the PR2 robot, and show its benefits over these approaches.**

## I. Introduction

Consider a typical mobile manipulation planning problem for a personal robot: the user requests the robot to fetch an object from a different room in the home. Not only is the configuration space of this problem high dimensional, but there are several 'narrow passages' in the configuration space—an example would be going through tight doors (Figure 1). The number of valid arm configurations for the robot when passing through the door is very small; most valid configurations will require the arms to be folded in. In such a scenario sampling-based motion planners suffer drastically, since the probability of sampling a valid configuration in the narrow passage, from the set of all possible configurations in the world is vanishingly small. Search-based planners on the other hand do not suffer from this problem, since they systematically expand the search graph, guided by a heuristic. However, if the heuristic were to have a deep local minimum, then the search-based planner spends a significant amount of time 'filling up' this local minimum or depression region. As an example, consider a typical heuristic used for mobile manipulation—the Dijkstra distance to goal for the base of the robot. While this heuristic guides the search away from locations where the base needs to travel a lot to get to the goal, it is completely uninformative for the other planning dimensions. At the doorway for instance, the heuristic provides no information on how to generate successor states for the arm. The result is that the planner

[†]Robotics Institute, Carnegie Mellon University, Pittsburgh, USA
fi@andrew.cmu.edu, {venkatraman,maxim} at cs.cmu.edu

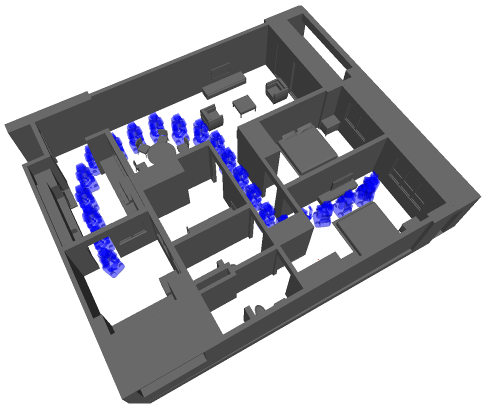

Fig. 1. Illustration showing a typical mobile manipulation problem where a personal robot has to fetch an object from a particular room in the home. Notice the several narrow corridors and tight spaces in the environment. Shown in blue is the plan generated by our proposed algorithm, Dynamic Multi-Heuristic A* (DMHA*).

ends up constructing a very dense graph at the doorway by trying every arm motion, eventually running out of time before it finds a state that allows it to get through the door.

The above describes a typical local minima problem for search-based planners. This effect is only aggravated when using methods such as Weighted A* [15], where the heuristic is inflated by some factor greater than 1 to create a greedy depth-first behavior. In this work, our primary contribution is a method to address the local minima problem by generating heuristics dynamically to guide the search out of depression regions. The intuition behind our method is that it is often easy to find a state in the vicinity of the local minimum, such that the found state is *not* in the depression region. Once we find this 'attractor state', we can generate a heuristic whose sole purpose is to lead the search towards the attractor state. Once the search gets out of the depression region, the regular heuristics can then take over and guide the search towards the goal in the usual manner.

Multi-Heuristic A* (MHA*) [1] is a recently proposed method that attempts to leverage the information provided by multiple (and possibly inadmissible) heuristics to circumvent local minima during search. Moreover, MHA* is able to provide guarantees on completeness and solution suboptimality bounds, despite using inadmissible heuristics. While the idea and solution quality bounds look promising, it requires the user to manually design heuristics that can each be independently useful in different parts of the state space. This is often a time-consuming process and requires the user to have excellent domain knowledge for generating good heuristics. In this work, we obviate the need for the user to carefully design local-minima free heuristics, by instead generating heuristics dynamically *on-the-fly* when necessary. By using our dynamic heuristic with the MHA* framework, we can reduce the engineering effort on the part of the user while at the same time providing all the theoretical guarantees that MHA* does.

## II. Related Work

The idea of automatically detecting local minima and guiding the search away from them has been explored in the past. R* search [13] is a randomized version of A* search that aims to circumvent local minima by generating random successors for a state, and then solving a series of short-range local planning problems on demand. Diankov and Kuffner propose a method called 'Randomized A*' [4], primarily for dealing with discretization issues in continuous state spaces. They also describe a statistical framework for learning heuristics from data to minimize engineering effort. A multi-heuristic search method is proposed in [7], that generates dynamic subgoals when the search is trapped in a local minimum. The primary difference between these methods and our proposed approach is that we do not require the search to expand the generated subgoal, or a random successor in the case of R*. Instead, the heuristic that we generate on the fly works in conjunction with the other baseline heuristics and attempts to guide those individual searches out of their local minima. Moreover, the proposed approach can provide deterministic guarantees on completeness and suboptimality bound of the solution returned, unlike the probabilistic guarantees of R*.

Agent-centric heuristic search methods form a different class of methods that handle dynamic heuristic updates. Most of these methods are based on Learning Real-Time A* [10], that interleaves planning and execution. Between the two phases, the algorithm updates the heuristic value of the expanded states, based on the heuristic values of the frontier states. While these methods attempt to improve the heuristic dynamically, they often need to update the heuristic value of states in the local minima multiple times before they can exit the depression region. On the other hand, the dynamic heuristic that we generate only attempts to guide the search out of the local minima, while not caring about the true cost-to-go value of the states.

The crux of our method depends on finding a state in the vicinity of a local minimum such that the found state itself is not in the local minimum region. A natural way to find this state is by doing rejection sampling in a region around the local minimum center. Sampling has been used with great success in motion planning for robots. RRT-Connect [11] and PRM [8] are two of the most popular motion planning algorithms that rely on random sampling of the configuration space. While the former grows two trees randomly out from the start and goal, the latter builds a roadmap generated by drawing random collision-free samples from the configuration space. While we also use sampling in our method, it is fundamentally different from the sampling-based family of algorithms in that the sampled state is used only for generating a new heuristic to guide the actual search. Methods described in in [9], [17] are complementary to ours in that they use search techniques to bias the growth of an RRT tree towards promising regions of the state-space.

## III. Background: Multi-Heuristic A*

Since we use our dynamic heuristic generation within the MHA* [1] framework, we first provide a brief summary of MHA* and its properties.

**Notations :** 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 $c(s, s') = \infty$. $\textsc{Succ}(s) := \{s' \in S | c(s, s') \neq \infty\}$, denotes the set of all successors of $s$. We use $c^*(s, s')$ to denote the cost of the optimal path from state $s$ to $s'$, $g(s)$ to denote the current best path cost from $s_{start}$ to $s$, and $h(s)$ to denote the heuristic for state $s$, which is an estimate of the best path cost from $s$ to $s_{goal}$. A heuristic is *admissible* if it never overestimates 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 $s' \in \textsc{Succ}(s)$ and $s \neq s_{goal}$. OPEN denotes a priority queue, and is typically implemented as a min-heap. Finally, we use $bp(s)$ to denote the backpointer for a state, i.e, its best parent state at any point during the search.

Optimal search algorithms such as A* [5] suffer in problems with large state-spaces because of the dramatic increase in the time and memory required. WA* [15] is a variant of A* that is often used to solve such large problems by trading-off optimality for speed. 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 [2], [19], [12]. WA* guarantees that the suboptimality of the solution is bounded by $w$ times the optimal cost [14] if $h(s)$ is admissible, and does not require re-expansions to guarantee the bound if $h(s)$ is consistent [12].

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 [6], [18] owing to its greedy nature. Multi-Heuristic A* (MHA*) [1] 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 inadmissible heuristics. MHA* uses multiple such (possibly inadmissible) heuristics to guide the search around local minima, by exploiting the synergy provided by these heuristics, each of which may be useful in different parts of the search space.

Formally, MHA* takes in one consistent heuristic ($h_0$), a set of arbitrary inadmissible heuristics ($h_1..h_n$) and two weight factors $w_1$ and $w_2$ (both $\geq 1$). It then runs multiple WA* searches (with weight $w_1$) 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

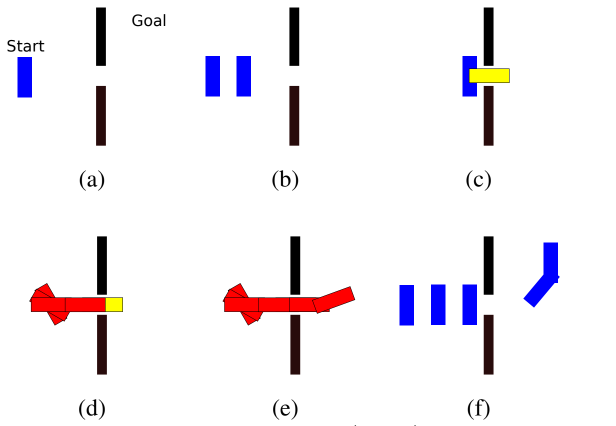

Fig. 2. Behavior of DMHA* in a holonomic $(x, y, \theta)$ planning problem. (a) Start and Goal positions. (b) Expansions (in blue) from the baseline heuristic queue. (c) The sampled attractor state (in yellow). (d) Expansions (in red) from the dynamic heuristic queue. (e) The dynamic heuristic queue makes its way out of the baseline heuristic depression region. (f) The baseline heuristic queue latches onto the state outside its depression region, and the dynamic queue is suspended.

inadmissible heuristics whose priority is within $w_2$ of the smallest priority in the open list of the anchor search (line 26 in Alg. 1). This way, MHA* can guarantee completeness and bounded suboptimality ($w_1 * w_2$) of the solution. In [1], two variants of MHA* were described, namely Independent Multi-Heuristic A* (IMHA*) which uses independent $g$ and $h$ values for each search and Shared Multi-Heuristic A* (SMHA*) which uses independent $h$ values but shares the $g$ value among all the searches. 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*. Tradeoffs between the two methods are discussed in [1].

## IV. Dynamic Multi-Heuristic A*

Our algorithm, Dynamic Multi-Heuristic A* (DMHA*) builds upon Shared Multi-Heuristic A* (SMHA), where the $g$-values are shared among all queues. We first convey the intuition and behavior of DMHA* with a simple example of a $(x, y, \theta)$ motion planning problem, shown in Figure 2.

In the figure, the rectangular holonomic robot has to navigate from the marked starting position to the goal position by passing through a narrow passage (Figure 2a). Let us assume that there is a simple baseline heuristic computed by running a 2D Dijkstra search starting from the goal. The heuristic for a state $(x, y, \theta)$ of the robot is then the Dijkstra distance from the cell $(x, y)$ to the goal. While this heuristic captures some information about obstacles in the environment, it does not account for the orientation of the robot. Since the initial state of the robot has an 'incorrect' orientation for passing through the narrow passage, the search gets stuck at the entrance to the narrow passage (expanding states without making progress toward the goal). This is a local minimum for the search as the heuristic is no longer decreasing and no longer providing guidance, causing the search to expand states uniformly around the state "closest" to the goal according to the baseline heuristic. Eventually, the search would fill up the entire depression region around the local minimum.

DMHA* monitors the progress of every baseline heuristic and creates a dynamic heuristic when all of the individual baseline heuristics are stuck in a local minimum. In the example, we have only one baseline heuristic search, shown in blue (Figure 2b). When the search is "stuck", DMHA* randomly samples a state in the vicinity of the local minimum such that the sampled state has a smaller baseline heuristic than the local minimum state. This is shown in Figure 2c, where a state with a smaller Dijkstra distance heuristic was sampled in the narrow passage. We call this state the attractor state ($s_{attractor}$), as we want the search to be pulled towards it, and away from the local minimum. A new heuristic is dynamically generated at this point which biases the search towards $s_{attractor}$. We then instantiate a new search queue containing all the states generated by the baseline heuristic, and update their priorities based on the dynamic heuristic. Figure 2d shows the state expansions from this dynamic queue. Since this queue contains all the states that were generated by the baseline heuristic, it directly expands that state which promises to align with $s_{attractor}$ the best. The dynamic search queue then immediately makes its way out of the baseline depression region (Figure 2e). Since the generated successors are shared between all the queues in SMHA*, the baseline search latches onto the state which is outside its depression region, and continues to make progress from thereon. Once any of the baseline searches start making progress, the dynamic queue is emptied and suspended until a new local minimum is encountered (Figure 2f).

### A. *Algorithm*

DMHA* is presented in Alg. 1. The colored lines show our contributed part, while the rest is the original SMHA* algorithm. As in SMHA*, the planner is provided with 1 consistent heuristic and $n$ (possibly inadmissible) baseline heuristics. The search proceeds in the usual SMHA* round-robin fashion, cycling between expansions from different queues. We additionally have one more OPEN list for the dynamic heuristic, called OPEN$_{dyn}$, which is always empty as long as one of the baseline searches is making progress.

The algorithm proceeds by monitoring each of the baseline heuristic searches to check for a local minimum. This is done as follows: whenever a state $s$ is expanded from OPEN$_i$, we check if its $h$-value is lower than the smallest $h$-value seen for a state expanded from OPEN$_i$ previously. The failure of this check indicates that heuristic $i$ has guided the search into a local minimum. When all OPEN$_i$, $i = 1, .., n$, and OPEN$_{dyn}$ (when its non-empty) are observed to be stuck in their respective local minima, OPEN$_{dyn}$ is reset with the states from OPEN$_i$ (line 36). Note that we can use any $i$ because all queues in SMHA* contain the exact same set of frontier states. Alg. 2 describes the InLocalMinima method that determines if all the baseline heuristic searches have entered a depression region. We generate $s_{attractor}$ in the vicinity of the local minima, subject to the condition that at least one of the baseline heuristics for that state

**Algorithm 1** DMHA*

```
1: procedure KEY($s, i$)
2:     return $g(s) + w_1 * h_i(s)$;
3:
4: procedure EXPAND($s$)
5:     Remove $s$ from OPEN$_i$ $\forall i = 0, \ldots, n$
6:     for each $s'$ in SUCC($s$) do
7:         if $s'$ was never visited then
8:             $g(s') = \infty$; $bp(s') = null$
9:         if $g(s') > g(s) + c(s, s')$ then
10:            $g(s') = g(s) + c(s, s')$; $bp(s') = s$
11:            if $s'$ has not been expanded in the anchor search then
12:                insert/update $(s')$ in OPEN$_0$ with $key(s', 0)$
13:                if $s'$ has not been expanded in any inadmissible search then
14:                    for $i = 1$ to $n$ do
15:                        if $key(s', i) \leq w_2 * key(s', 0)$ then
16:                            insert/update $(s')$ in OPEN$_i$ with $key(s', i)$
17:
18: procedure MAIN()
19:     $g(s_{goal}) = \infty$; $bp(s_{start}) = bp(s_{goal}) = null$
20:     $g(s_{start}) = 0$
21:     for $i = 0$ to $n$ do
22:         OPEN$_i$ = $\emptyset$
23:         insert $s_{start}$ into OPEN$_i$ with $key(s_{start}, i)$ as priority
24:     while OPEN$_0$ not empty do
25:         for $i \in \{1, .., n\} \cup dyn$ do
26:             if OPEN$_i.Minkey() \leq w_2 \cdot$ OPEN$_0.Minkey()$ then
27:                 if $g(s_{goal}) \leq$ OPEN$_i.Minkey()$ then
28:                     terminate and return path pointed by $bp(s_{goal})$
29:                 $s$ = OPEN$_i.Top()$
30:                 EXPAND($s$)
31:                 if INLOCALMINIMA() then
32:                     $s_{attractor} \leftarrow$ SAMPLEATTRACTOR()
33:                     if $s_{attractor}$ = NULL then
34:                         continue // return control to line 25
35:                     else
36:                         OPEN$_{dyn}$ $\leftarrow$ COPY(OPEN$_i$)
37:                         $h_{dyn}(s') = \Delta(s', s_{attractor}) \forall s' \in$ OPEN$_{dyn}$
38:                         UPDATEPRIORITIES(OPEN$_{dyn}$)
39:                 else
40:                     OPEN$_{dyn}$ $\leftarrow \emptyset$
41:             else
42:                 if $g(s_{goal}) \leq$ OPEN$_0.Minkey()$ then
43:                     terminate and return path pointed by $bp(s_{goal})$
44:                 $s$ = OPEN$_0.Top()$
45:                 EXPAND($s$)
```

is smaller than the best seen for the corresponding queue (line 37 and 38). The priorities of the states in OPEN$_{dyn}$ are then recomputed according to the attractor heuristic, which is simply a domain dependent distance metric ($\Delta$) between a given state and $s_{attractor}$. Note that $s_{attractor}$ does not act as a sub-goal, but only pulls the expansions towards itself, for the duration needed for one of the baseline searches to escape their local minimum. This is due to the shared nature of the algorithm, where a successor generated by OPEN$_{dyn}$ is copied to all other OPEN$_i$. It could be the case that there is no $s_{attractor}$ or the sampling takes prohibitively long (greater than $T_{sampling}$), in which case the SAMPLEATTRACTOR method returns NULL, and we continue executing the usual round-robin SMHA* search with the baseline heuristics. The details and complexity of sampling $s_{attractor}$ are in general domain-specific. However, for most robotics problems such as mobile manipulation, this is straightforward. The experiments section describes the sampling procedure we use in greater detail.

**Algorithm 2** DMHA* Procedures

```
1: procedure INLOCALMINIMA()
2:     for $i \in \{1, .., n\} \cup dyn$ do
3:         $\mathcal{S} \leftarrow$ set of all states expanded from OPEN$_i$
4:         $s \leftarrow$ state last expanded from OPEN$_i$
5:         $s' \leftarrow \mathrm{argmin}_{\mathcal{S}}\, h_i(s)$
6:         if $h_i(s) < h_i(s')$ then
7:             return false
8:     return true
9: procedure SAMPLEATTRACTOR()
10:    for $i \in \{1, .., n\}$ do
11:        $\mathcal{S}_i \leftarrow$ set of all states expanded from OPEN$_i$
12:        $s_i \leftarrow \mathrm{argmin}_{\mathcal{S}_i}\, h_i(s)$
13:    $s_{attractor} \leftarrow$ sampled state $s$ such that
                $h_i(s) < h_i(s_i)$ for at least one $i$
14:    if $s_{attractor}$ found within time $T_{sampling}$ then
15:        return $s_{attractor}$
16:    else
17:        return NULL
```

## B. Theoretical Properties

*Theorem 1:* The cost of the solution returned by Dynamic Multi-Heuristic A* is bounded by $w_1 * w_2$ of the optimal solution cost.

The proof of this follows from the properties of SMHA*. SMHA* can take in a number of inadmissible heuristics and still provide a solution quality bound as long as there is one consistent heuristic [1] used for the 'anchor' search. Since our method runs under the SMHA* framework, it does not matter if the dynamically generated heuristic is inadmissible. Moreover, removing or adding a new heuristic does not affect any of the operations of the anchor search, thereby permitting us to dynamically add a search queue when necessary.

*Theorem 2:* Dynamic Multi-Heuristic A* is complete with respect to the underlying search graph.

While the original SMHA* method is complete, one might be concerned that the probabilistic sampling component of DMHA* might affect its completeness. However, since we always timeout when $s_{attractor}$ is not generated within a particular time, the search falls back to round-robin expansions using the baseline heuristics. This guarantees completeness.

## V. Experimental Results

We evaluated the performance of DMHA* on a 12 DOF mobile manipulation problem for the PR2 robot. The PR2 robot has two arms with 7 joints each, a holonomic base and a prismatic spine. The state space representation which we used for planning is similar to Cohen et al. [3]. The robot configuration is represented by a 12 DOF pose: a 6 DOF object pose for the end-effector, 2 redundant arm joints, $(x, y, \theta)$ for the base, and the prismatic spine height. The start state for the robot is represented by the full body configuration which includes all the 12 dimensions. The goal state is represented by a 6 DOF pose $(x, y, z, roll, pitch, yaw)$ for the end-effector and is hence underspecified. The environment we used for our experiments is a typical home environment with multiple rooms, doorways and corridors (Figure 3). The search graph for the problem is constructed on the fly, by applying a set of 'motion primitives' to the state being expanded. These motion primitives are small kinematically feasible motions that the robot can execute.

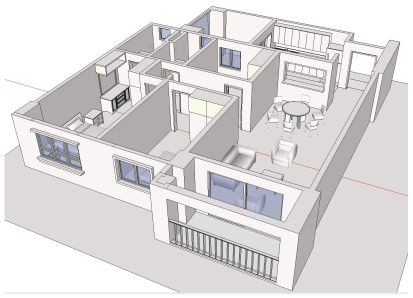

Fig. 3. The house environment model used for experiments.

An example would be a motion primitive that only moves the base forward by 20cm in the x-direction, while leaving all other dimensions unchanged.

## A. Heuristics

For our baseline heuristics ($h_1$ and $h_2$), we use two common and easy to compute ones, namely the base heuristic and the end-effector heuristic [1]. Both of these heuristics are admissible and they are calculated as follows: the end-effector heuristic is computed as the Euclidean distance in $(x, y, z)$ from the end-effector position to the goal position. The base heuristic is calculated by running a 2D Dijkstra search for the robot base for which the goal region is defined by a circle centered around the $(x, y)$ projection of the goal pose. The purpose of this circular region is to maintain an admissible heuristic despite having an underspecified search goal. As the set of possible goal states must have the robot base within arm's reach of the goal, we ensure that the heuristic always underestimates the actual cost to goal by setting the radius of the circular region to be slightly larger than the maximum reach of the robot arm. Finally, we use the base heuristic as the anchor heuristic, $h_0$, since it is consistent and admissible.

For computing the dynamic heuristic, we only need to define a domain-specific distance metric for a pair of states. Here, we simply use the Euclidean distance in the full 12 DOF configuration of the robot, with an additional detail that every dimension is scaled according to the minimum cost incurred by an action along that dimension. For instance, if moving the base 20cm along the x-direction has a cost of 100 units in the graph (the corresponding motion primitive has a cost of 5 units/cm), then the x-component of our Euclidean distance is scaled by 5. This helps translate configuration space distances to costs on the search graph and puts the heuristic on the same scale as the $g$ values.

## B. Sampling an Attractor State

When the baseline heuristic searches are stuck in local minima, the algorithm requires us to sample an attractor state in the vicinity of one of the local minima centers, such that it has a smaller heuristic value. For the mobile manipulation problem, we do this by sampling a random 12 DOF configuration from a union of 12D hyperellipsoids centered at the local minima of all baseline heuristic searches, and reject those samples that do not satisfy the condition on the heuristic. Alternatively, one could repeatedly apply a sequence of random motion primitives starting from the local minima states, until a resulting state is found that satisfies the attractor condition.

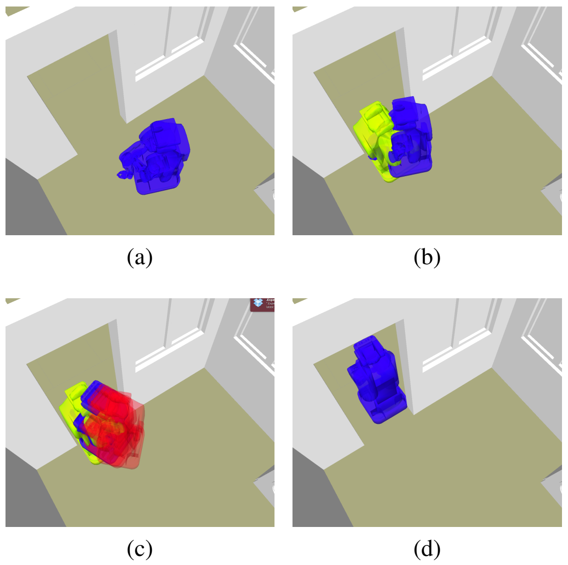

Fig. 5. Example showing the operation of the dynamic heuristic. Here, the problem is to find a plan that takes the PR2 through the doorway. (a) The starting configuration expanded from the base heuristic search queue (in blue). (b) A local minimum state for the base heuristic search (blue) and the sampled attractor state (yellow). (c) Expansions from the dynamic heuristic search (in red) (d) The base heuristic search escapes the local minimum, and the dynamic heuristic search is reset.

Note that there could be a scenario where all the baseline heuristics have reached their respective 'goals', i.e, all the baseline heuristics are identically 0. This typically happens when the robot has reached a point on the circle around the goal pose, and the end-effector has its $(x, y, z)$ to be the same as the goal $(x, y, z)$, but with different roll, pitch and yaw. This is now a local minimum for the baseline heuristics, and it is not possible to find an attractor state with a baseline heuristic smaller than 0. In such a scenario, we instead sample from the manifold of valid goal configurations for the robot by generating a base configuration $(x, y, \theta)$ from which there exists a valid inverse kinematics solution to the goal object pose. The sampling scheme that is described is only a strawman proposal to help demonstrate the effectiveness of DMHA*. In practice, one could use additional domain knowledge or a more sophisticated sampling method to help accelerate finding the attractor state.

Figure 5 shows the process of generating an attractor state. Here, the robot needs to find a plan that takes it from the starting location (Figure 5a), to a goal on the other side of the door. Figure 5b shows a state (in blue) expanded from the base heuristic search. This is a local minimum state because the arm configuration is such that the robot cannot pass through the doorway. The algorithm then immediately generates an attractor state (shown in yellow, Figure 5b). Note that the base heuristic for this state is better than that of the local minimum state, since it is "closer" to the goal according to the base heuristic. Once the attractor state is generated, the dynamic queue is created and it starts expanding states (shown in red, Figure 5c),

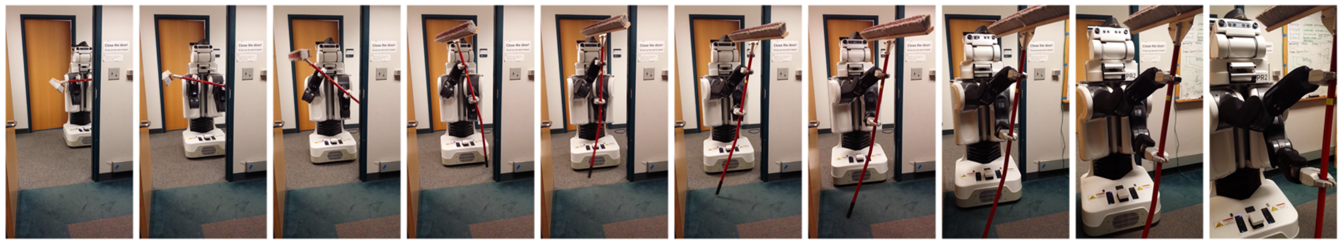
Fig. 4. The PR2 robot executing a full-body manipulation task that requires moving a broom from the hallway to a room. Note that the planner has to jointly reason about moving the robot's base and the arms so that the object can fit through the doorway.

which drive it towards the attractor. At the same time, the base heuristic search also continues its regular operation, expanding states in the local minimum. Soon, the dynamic heuristic search expands a state closely aligned with the attractor state. Because the successor states are shared among all the queues, the base heuristic search now immediately expands this state (because it has a smaller heuristic than its local minimum heuristic), and continues to make progress from thereon (Figure 5d). At this point, the dynamic heuristic search is reset and suspended.

### C. Mobile Manipulation on the PR2 Robot

We tested our algorithm DMHA* on the physical PR2 robot for a full-body mobile manipulation task. The task assigned to the robot was to carry a large T-shaped broom through the hallway and then into a room. Note that this problem is challenging in several aspects: the planner has to jointly reason about moving the base and its arms because of the tight passages and the large object that it is carrying. Consequently, simple heuristics such as the base and end-effector heuristics described earlier are insufficient for this problem as they cannot provide information to guide the search in the full 12 DOF space. As a result SMHA* with these simple heuristics fails to find a solution. On the contrary, DMHA* generates appropriate dynamic heuristics whenever necessary to get the search out of the local minima for the simple heuristics, and is able to find a solution within 60 seconds. The attached video shows the robot executing the plan while Figure 4 provides some screenshots from the video.

### D. Simulation Results and Comparisons

For comprehensive evaluation, we compared DMHA* against the original SMHA* method and popular sampling-based algorithms including RRT-Connect, PRM, KPIECE and EST from the Open Motion Planning Library (OMPL) [16]. Unlike the real PR2 experiment which involved a dual-arm mobile manipulation task, we used single-arm mobile manipulation for our simulation experiments. We generated 100 random start-goal pairs in the home environment and ran each of the methods on those instances. For all the methods the start state was a full 12 DOF configuration of the robot. For SMHA* and DMHA*, the goal was an under-specified 6 DOF end-effector pose $(x, y, z, roll, pitch, yaw)$, while for RRT-Connect, the goal state was a full 12 DOF

TABLE I
COMPARISON OF DMHA* WITH SMHA* AND RRT-CONNECT

| | SMHA* | DMHA* |
|---|---|---|
| Success Rate(percent) | 25 | 82 |
| Mean planning time(s) | 6.58 | 4.76 |
| Mean state expansions | 465 | 399 |
| Mean path length | base 5.99m, arms 7.11rad | base 6.06m, arms 7.34rad |

(a) SMHA* and DMHA*

| | RRT-Connect | DMHA* |
|---|---|---|
| Success Rate(percent) | 29 | 82 |
| Mean planning time(s) | 10.15 | 3.79 |
| Mean path length | base 8.0m, arms 35.5rad | base 5.4m, arms 7.7rad |

(b) RRT-Connect and DMHA*

configuration that satisfied the same 6 DOF end-effector pose. For SMHA*, we used the same baseline and anchor heuristics as that used by DMHA*. Also, for both the MHA* methods we used a value of 25 for $w_1$ and 4 for $w_2$. Since our evaluation is only on single-arm mobile manipulation tasks, we constrain the attractor state sampling for DMHA* by generating only those samples in the vicinity of the robot that have the arms folded in towards the body, i.e, the sampling is effectively only in $(x, y, \theta)$ of the robot's base. This helps reduce the number of rejections needed before a valid attractor state can be sampled as rejection sampling in high-dimensional configuration spaces is expensive. All experiments were performed on an Intel i7 - 3770(3.4GHz) PC with 16GB RAM, and the timeout for planning was set to 60 seconds. We declare a trial as successful if a solution was returned before the timeout. Of all the sampling-based methods we compared with, only RRT-Connect achieved a success rate of over 10% and therefore we report detailed comparisons only with RRT-Connect.

Table I compares the performance of DMHA* with SMHA* and RRT-Connect. The algorithms are compared against the measures of mean planning time, mean number of expansions (only for SMHA* and DMHA*), and the mean path length computed in terms of the total distance traveled by the base (in meters), and the sum angular distance covered by all the joints (in radians). The mean statistics are computed only on common trials in which DMHA* and the method being compared to, both succeeded. Our proposed method shows substantial improvements in the success rate as compared to both SMHA* and RRT-Connect. We are able to solve over 50% more problems than either

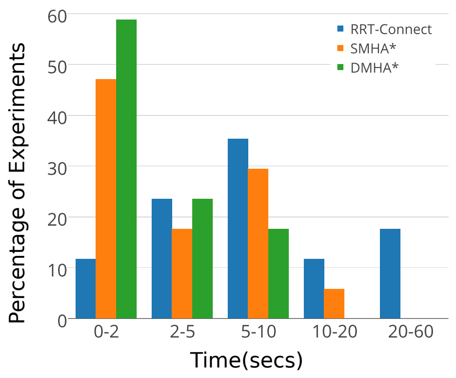

Fig. 6. Planning time distributions for RRT-Connect, SMHA* and DMHA*. The planning times used for this computation are only from trials on which all three methods succeeded.

of these methods. This follows from the fact that SMHA* did not have enough carefully designed baseline heuristics to help it avoid local minima, while RRT-Connect simply cannot handle the large number of narrow passages in the configuration space. The mean planning time for DMHA* is only slightly better than that for SMHA*. This is expected, since the majority of problems solved by SMHA* are the ones that have no local minima. In those cases DMHA* behaves exactly the same as SMHA*. The mean planning time for RRT-Connect is much larger however, since it takes significant time to sample configurations in narrow passages.

The histogram in Figure 6 shows the distributions of planning times for RRT-Connect, SMHA* and DMHA*, computed over planning trials where all three methods were successful. While the distribution for DMHA* is skewed towards the left, SMHA* and RRT-Connect both have centered distributions. In particular, DMHA* found a solution within 2 seconds in 58% of the common successful trials. This demonstrates that DMHA* not only provides better success rates but also better planning times.

The number of expansions and average path lengths are more or less identical for SMHA* and DMHA*. This is expected since they are both search-based methods that explicitly optimize for solution quality. On the other hand, the average path lengths for RRT-Connect are significantly worse (even after running a short-cutter on the final path), especially for the joint angle motions.

## VI. Conclusions

We presented DMHA*, an algorithm that dynamically generates heuristics to avoid local minima in search-based planning. This relieves the user from manually engineering heuristics based on the domain to help improve search performance. The dynamically generated heuristic is used under the Multi-Heuristic A* framework, where search information is shared between multiple search queues. From a theoretical side, DMHA* provides guarantees on completeness and bounds on suboptimality of the solution found. Our experiments on full-body mobile manipulation for the PR2 robot show that DMHA* outperforms a naive version of SMHA* that uses only simple heuristics designed without much effort, and RRT-Connect, a popular sampling-based motion planning algorithm. In particular, our success rate for a typical home environment was significantly better than the other methods, showing that DMHA* is an ideal candidate for personal robot mobile manipulation planning.

An interesting direction for future work would be to leverage more sophisticated sampling schemes to accelerate finding the attractor state. Additionally, we would like to extend our method to an anytime version, where the solution quality can be improved as time permits.

## Acknowledgments

This research was sponsored by the ONR DR-IRIS MURI grant #N00014-09-1-1052. The authors would like to thank Mike Phillips for fruitful discussions.

## References

[1] 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.

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

[3] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for dual-arm manipulation with upright orientation constraints. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3784–3790. IEEE, 2012.

[4] Rosen Diankov and James Kuffner. Randomized statistical path planning. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1–6. IEEE, 2007.

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

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

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

[8] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[9] Scott Kiesel, Ethan Burns, and Wheeler Ruml. Abstraction-guided sampling for motion planning. In *SOCS*, 2012.

[10] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2):189–211, 1990.

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

[12] 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.

[13] Maxim Likhachev and Anthony Stentz. R* search. *Lab Papers (GRASP)*, page 23, 2008.

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

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

[16] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012.

[17] Vojtěch Vonásek, Jan Faigl, Tomáš Krajník, and Libor Přeučil. Rrt-path–a guided rapidly exploring random tree. In *Robot Motion and Control 2009*, pages 307–316. Springer, 2009.

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

[19] 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.