Multi-Heuristic A*

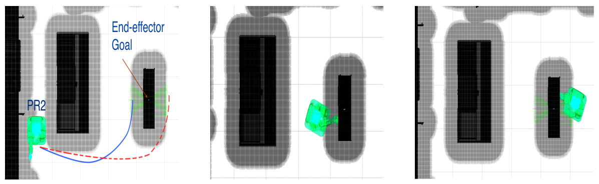
Sandip Aine*, Siddharth Swaminathan†, Venkatraman Narayanan†, Victor Hwang† and Maxim Likhachev†
*Indraprastha Institute of Information and Technology, New Delhi, India
†Carnegie Mellon University, Pittsburgh, PA, USA

***Abstract*—**
The performance of heuristic search (such as A*) based planners depends heavily on the quality of the heuristic function used to focus the search. These algorithms work fast and generate high-quality solutions, even for high-dimensional problems, as long as they are given a well-designed heuristic function. Consequently, the research in developing an efficient planner for a specific domain becomes the design of a good heuristic function. However, for many domains, it is hard to design a *single* heuristic function that captures all the complexities of the problem. Furthermore, it is hard to ensure that heuristics are admissible and consistent, which is necessary for A* like searches to provide guarantees on completeness and bounds on suboptimality. In this paper, we develop a novel heuristic search, called Multi-Heuristic A* (MHA*), that takes in *multiple, arbitrarily inadmissible* heuristic functions in addition to a single consistent heuristic, and uses all of them simultaneously to search for a complete and bounded suboptimal solution. This simplifies the design of heuristics and enables the search to effectively combine the guiding powers of different heuristic functions. We support these claims with experimental analysis on several domains ranging from inherently continuous domains such as full-body manipulation and navigation to inherently discrete domains such as the sliding tile puzzle.

I. Introduction

A* search [10] has been used abundantly for low-dimensional path planning in robotics since 1980s. Within the last decade, it has also been shown that suboptimal versions of A* such as Weighted A* (WA*) [24] and its anytime variants [19, 31], can also be used quite effectively for high-dimensional planning problems in robotics ranging from motion planning for ground vehicles [20] and flight planning for aerial vehicles [21] to planning for manipulation [7] and footstep planning for humanoids [12] and quadrupeds [32]. All of these planners achieve faster speeds than A* search by inflating the heuristic values with an inflation factor ($w > 1$) to give the search a depth-first flavor. As such though, they rely heavily on the guiding power of the heuristic function. In fact, WA*'s performance can degrade severely in the presence of heuristic depression regions, i.e., regions in the search space where the correlation between the heuristic values and the actual cost of the path to goal is weak [11, 30]. WA* can easily get trapped in these regions as its depth-first greediness is guided by the heuristic values, and may require expanding most of the states belonging to a depression zone before exiting. Designing a powerful heuristic function that is also valid, namely admissible and consistent, is therefore difficult for complex planning problems.

This research was sponsored by the ONR DR-IRIS MURI grant N00014-09-1-1052 and DARPA Computer Science Study Group (CSSG) grant D11AP00275.



(a) Heuristics (b) Admissible Search (c) Inadmissible Search

Fig. 1: A full-body (11D: x,y,orientation for the base + spine + 7D arm) manipulation planning (with PR2) example depicting the utility of multiple heuristics. Figure 1a shows two heuristic paths that corresponds to greedily following two different heuristics, blue (solid) curve for an admissible heuristic and red (dotted) curve for an inadmissible heuristic. The admissible heuristic guides the search to a depression region where the search gets stuck (Figure 1b). In contrast, the inadmissible heuristic guides the search along a feasible path and therefore allows the planner to find a valid plan fast (Figure 1c).

In contrast, for many domains, one can easily compute different inadmissible heuristics, each of which can provide complementary guiding power. For example, in Figure 1 we include a full-body manipulation scenario where the admissible heuristic function (path shown by the blue solid curve, Figure 1a) guides the search to a local minimum as the robot cannot reach the object from the left side of the table (Figure 1b). However, we can obtain multiple inadmissible heuristics by computing the navigation (x, y) distance to different points around the object to be grasped. In the example (Figure 1a), we show one such additional heuristic function that guides the search through a different route (shown by the red dotted curve to the right side of the table). Using this heuristic, the search does find a pose that allows the robot to grasp the object, i.e., it computes a valid plan (Figure 1c).

When used in isolation, these additional heuristics provide little value, because they can neither guarantee completeness (because they can be arbitrarily inadmissible), nor guarantee fast planning times (because each may have its own depression region). We show in this paper that a search can consider multiple such heuristics to explore different parts of the search space while using a consistent heuristic to ensure completeness. This may result in faster convergence, if any of these heuristics (or their combination) can effectively guide the search around the depression regions.

We present an algorithmic framework, called Multi-Heuristic A* (MHA*), which works by running multiple searches with different inadmissible heuristics in a manner that preserves completeness and guarantees on the suboptimality bounds. We propose two variants of MHA*: Independent

Multi-Heuristic A* (IMHA*) which uses independent g and h values for each search, and Shared Multi-Heuristic A* (SMHA*) which uses different h values but a single g value for all the searches. We show that with this shared approach, SMHA* can guarantee the suboptimality bounds with at most two expansions per state. In addition, SMHA* is potentially more powerful than IMHA* in avoiding depressions as it can use a combination of partial paths found by different searches to reach the goal. We discuss the theoretical properties of MHA* algorithms stating their completeness, suboptimality bounds and complexities (in terms of state expansions). We present experimental results for the following robotics domains: 12D mobile manipulation for PR2 (full-body) and 3D (x, y, orientation) navigation. These experiments demonstrate the efficacy of MHA*, especially for cases where the commonly used heuristics do not lead the search well. We also include experimental results for large sliding tile puzzle problems, highlighting the benefits of the proposed framework outside robotics.

II. Related Work

The utility of having multiple heuristics is noted in many search applications including motion planning [20], searching with pattern database heuristics [9, 17], AND/OR graphs [6] etc. For example, in Likhachev and Ferguson [20] the maximum of two admissible heuristics (one mechanism-relative and another environment-relative) was used, as it could guide the planner better when compared to the individual heuristics. The key difference between these approaches and ours is that while these utilize multiple heuristics, the information is combined to create a single best (often consistent) heuristic to guide the search, whereas MHA* uses multiple heuristics independently to explore different regions of the search space. Also, as we derive the bounds using a consistent heuristic, the additional heuristics can be arbitrarily inadmissible.

The idea of deriving the bounds based on a consistent heuristic and using inadmissible estimates/constraints to guide the search has also been used in several other search algorithms [2, 5, 23, 27, 28]. For example, Explicit Estimation Search [27] uses an inadmissible distance function to guide the search, but derives the bounds using an admissible heuristic. While MHA* follows a similar approach to derive the bounds, the concept of simultaneous searching with different heuristics sets it apart.

In Röger and Helmert [25], an empirical examination was performed on how to exploit multiple heuristics for satisficing planning. Among the different benchmarked approaches, the alternation method was shown to be the best. This approach uses multiple heuristics independently (in contrast to combining them as sum, max etc) to explore the search space in a greedy best-first manner [22] while sharing the g values. In Isto [13], a robotics path planning algorithm was proposed that utilizes multiple heuristics and attempts to share the resource among individual searches. The basic philosophy of these algorithms is close to the MHA* approach, however, they do not provide any guarantees on completeness/suboptimality. In contrast, MHA* guarantees a) completeness, b) suboptimality and c) bounded expansions by using a consistent heuristic search to anchor the explorations.

Valenzano et al. [29] proposed simultaneous searching with different parameters (such as operator orders, heuristic weights), as an alternative to fine tune the parameters. This method was shown to be effective for several problems especially when resources are available for parallel exploration. However, this framework also relies on a single heuristic to guide the search (albeit with different parameters) and therefore can suffer similarly in presence of large depression regions.

A completely different approach to path planning is adopted by the sampling based planners [15, 16, 18]. The fundamental difference between the sampling and heuristic search based planners is that the sampling based algorithms primarily target continuous spaces whereas the search algorithms are for planning in discrete spaces, independent of whether these came as the result of discretizing the state-space or from an inherently discrete system. Moreover, heuristic search based planning methods often provide better cost minimization and more consistent behavior compared to the sampling based planners, but at the expense of higher planning times and need for a well-designed heuristic function. Our work targets the last issue as we try to alleviate the dependency on having a single well-designed heuristic function by supporting multiple heuristic functions that can be arbitrarily inadmissible.

III. Multi-Heuristic A*

In this section, we describe two multi-heuristic search algorithms and discuss their properties.

Notations and Assumptions : 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 $c(s, s') = \infty$. $Succ(s) := \{s' \in S | c(s, s') \neq \infty\}$, 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 optimal path cost from s_{start} to s. $g(s)$ denotes the current best path cost from s_{start} to s, and $bp(s)$ denotes a backpointer which points to the best predecessor of s (if computed).

We assume that we have n heuristics denoted by h_i for $i = 1..n$. These heuristics do not need to be consistent, in fact, they can be arbitrarily inadmissible. We also require access to a consistent (and thus admissible) heuristic (denoted by h_0), i.e., h_0 should satisfy, $h_0(s_{goal}) = 0$ and $h_0(s) \leq h_0(s') + c(s, s')$, $\forall s, s'$ such that $s' \in Succ(s)$ and $s \neq s_{goal}$. MHA* uses separate priority queues for each search ($n+1$ queues), denoted by $OPEN_i$, for $i = 0..n$. Throughout the rest of the paper, we will use the term *anchor* search to refer to the search that uses h_0. Other searches will be referred to as *inadmissible* searches.

A. Independent Multi-Heuristic A* (IMHA*)

The algorithm IMHA* is presented in Figure 2. In IMHA*, different heuristics are explored *independently* by simultaneously running $n + 1$ searches each using its own priority queue. Therefore, in addition to the different h values, each

```
1  procedure key($s, i$)
2  return $g_i(s) + w_1 * h_i(s)$;
3  procedure Expand($s, i$)
4  Remove $s$ from $OPEN_i$;
5  for each $s'$ in Succ($s$)
6   if $s'$ was never visited in the $i^{th}$ search
7    $g_i(s') = \infty$; $bp_i(s') = \textbf{null}$;
8   if $g_i(s') > g_i(s) + c(s, s')$
9    $g_i(s') = g_i(s) + c(s, s')$; $bp_i(s') = s$;
10   if $s'$ has not been expanded in the $i^{th}$ search
11     insert/update ($s'$) in $OPEN_i$ with $key(s', i)$;
12 procedure IMHA*()
13 for $i = 0$ to $n$
14  $g_i(s_{goal}) = \infty$; $bp_i(s_{start}) = bp_i(s_{goal}) = \textbf{null}$;
15  $g_i(s_{start}) = 0$; $OPEN_i = \emptyset$;
16  insert $s_{start}$ into $OPEN_i$ with $key(s_{start}, i)$ as priority;
17 while $OPEN_0$ not empty
18  for $i = 1$ to $n$
19   if $OPEN_i.Minkey() \leq w_2 * OPEN_0.Minkey()$
20    if $g_i(s_{goal}) \leq OPEN_i.Minkey()$
21     terminate and return path pointed by $bp_i(s_{goal})$;
22    $s = OPEN_i.Top()$;
23    Expand($s, i$);
24   else
25    if $g_0(s_{goal}) \leq OPEN_0.Minkey()$
26     terminate and return path pointed by $bp_0(s_{goal})$;
27    $s = OPEN_0.Top()$;
28    Expand($s, 0$);
```

Fig. 2: Independent Multi-Heuristic A* (IMHA*)

state uses a different g (and *bp*) value for each search. We use g_0 to denote the g for the *anchor* search, and g_i for the other searches. The suboptimality bound is controlled using two variables, namely, $w_1(\geq 1.0)$ which is used to inflate the heuristic values for each of the searches, and $w_2(\geq 1.0)$ which is used as a factor to prioritize the inadmissible searches over the *anchor* search. IMHA* runs the inadmissible searches in a round robin manner in a way that guarantees solution quality within the suboptimality bound of $w_1 * w_2$.

IMHA* starts with initializing search variables (lines 14-16) for all the searches. It then performs best-first expansions in a round robin fashion from queues $OPEN_i$ $i = 1..n$, as long as $OPEN_i.Minkey() \leq w_2 * OPEN_0.Minkey()$ (line 19). If the check is violated for a given search, it is suspended and a state from $OPEN_0$ is expanded in its place. This in turn can increase $OPEN_0.Minkey()$ and thus re-activate the suspended search. Expansion of a state is done in a similar way as done in A*. Each state is expanded at most once for each search (line 10) following the fact that WA* does not need to reexpand states to guarantee the suboptimality bound [19]. IMHA* terminates successfully, if any of the searches have $OPEN_i.Minkey()$ value greater than or equal to the g value of s_{goal} (in that search), otherwise it terminates with no solution when $OPEN_0$ is empty.

1) IMHA Properties:* In [3], we present detailed proofs for a number of properties of IMHA*. Here we briefly discuss the most important of those theorems. First, we note that the *anchor* search in IMHA* is a single shot WA* (without reexpansions) with a consistent heuristic function h_0. Thus, all the results of such a WA* are equally applicable in case of the *anchor* search in IMHA*. Next, we present two key theorems for the *anchor* search which shall later be used to derive the properties of IMHA*.

Theorem 1. *At line 18, for any state s with $key(s, 0) \leq key(u, 0) \forall u \in OPEN_0$, it holds that $g_0(s) \leq w_1 * g^*(s)$.*

Proof: We borrow this Theorem from the results described in [19], which states that the g value (g_0 here) for any state to be expanded in WA* (*anchor* search here) is at most w_1-suboptimal. ■

Theorem 2. *At line 18, for any state s with $key(s, 0) \leq key(u, 0) \forall u \in OPEN_0$, it holds that $key(s, 0) \leq w_1 * g^*(s_{goal})$.*

Proof: We prove this by contradiction. Let us assume, $key(s, 0) = g_0(s) + w_1 * h_0(s) > w_1 * g^*(s_{goal})$.

Let us consider a least cost path from s_{start} to s_{goal} given as $P = \Pi(s_0 = s_{start}, ..., s_k = s_{goal})$. From this path, we pick the first state s_i that has not yet been expanded by the *anchor* search, but is part of $OPEN_0$ ($s_i \in OPEN_0$). Note that we will always find such a state $s_i \in OPEN_0$ because a) $s_0 = s_{start}$ is put in $OPEN_0$ at the initialization (line 16), b) whenever any state $s_j \in P$ is expanded in the *anchor* search $s_{j+1} \in P$ is always inserted in $OPEN_0$, and c) $s_k = s_{goal}$ is never expanded in the *anchor* search, otherwise, whenever s_{goal} has the least key in $OPEN_0$ the search terminates (line 25).

Now, let us examine $g_0(s_i)$. If $i = 0$, we have $g_0(s_i) = g_0(s_{start}) = 0 \leq w_1 * g^*(s_i)$ (as, $g^*(s_{start}) = g_0(s_{start}) = 0$). If $i \neq 0$, by the choice of s_i we know that s_{i-1} has already been expanded in the *anchor* search. When s_{i-1} was chosen for expansion, we had $g_0(s_{i-1}) \leq w_1 * g^*(s_{i-1})$ from Theorem 1. Now, as s_i is a successor of s_{i-1}, we have

$$\begin{aligned} g_0(s_i) &\leq g_0(s_{i-1}) + c(s_{i-1}, s_i) \text{ (line 9, Figure 2)} \\ &\leq w_1 * g^*(s_{i-1}) + c(s_{i-1}, s_i) \\ &\leq w_1 * (g^*(s_{i-1}) + c(s_{i-1}, s_i)) \\ &\quad \text{As } s_{i-1}, s_i \in \text{optimal path,} \\ &= w_1 * g^*(s_i) \end{aligned} \tag{1}$$

Thus, we have $g_0(s_i) \leq w_1 * g^*(s_i)$. Using this we obtain,

$$\begin{aligned} key(s_i, 0) &= g_0(s_i) + w_1 * h_0(s_i) \\ &\leq w_1 * g^*(s_i) + w_1 * h_0(s_i) \\ &\leq w_1 * g^*(s_i) + w_1 * c^*(s_i, s_{goal}) \\ &\quad h_0 \text{ is consistent, thus admissible} \\ &= w_1 * g^*(s_{goal}) \end{aligned} \tag{2}$$

Now, as $s_i \in OPEN_0$ and $key(s_i, 0) \leq w_1 * g^*(s_{goal}) < key(s, 0)$, we have a contradiction to our assumption that $key(s, 0) \leq key(u, 0), \forall u \in OPEN_0$. ■

Next, we present three theorems summarizing the main properties of IMHA*.

Theorem 3. *When IMHA* exits (in the i^{th} search), $g_i(s_{goal}) \leq w_1 * w_2 * g^*(s_{goal})$, i.e., the solution cost obtained is bounded by $w_1 * w_2$ suboptimality factor.*

Proof: IMHA* can terminate successfully in lines 25 (*anchor search*) or 20 (inadmissible search), or it can terminate without a solution in line 17.

If the *anchor* search terminates at line 25, i.e., $key(s_{goal}, 0) \leq key(u, 0)$, $\forall u \in OPEN_0$, from Theorem 1 we have,

$$\begin{aligned} g_0(s_{goal}) &\leq w_1 * g^*(s_{goal}) \\ &\leq w_1 * w_2 * g^*(s_{goal}) \\ &\quad \text{As } w_2 \geq 1.0 \end{aligned} \tag{3}$$

If an inadmissible search (say i^{th}) terminates in line 20, then from lines 19 and 20, we have,

$$\begin{aligned} g_i(s_{goal}) &\leq w_2 * OPEN_0.Minkey() \\ &\leq w_2 * w_1 * g^*(s_{goal}) \\ &\text{From Theorem 2} \end{aligned} \tag{4}$$

Therefore, in both the above mentioned cases, we have the solution cost to be within $w_1 * w_2$ factor of the optimal solution cost. On the other hand, if the search terminates unsuccessfully at line 17 (while condition is not satisfied), from Theorem 2 we know $OPEN_0.Minkey() \leq w_1 * g^*(s_{goal})$. $OPEN_0 = \emptyset$ denotes $OPEN_0.Minkey() = \infty \implies g^*(s_{goal}) \geq \infty$, i.e., there is no finite cost solution. ■

Theorem 4. *No state is expanded more than* $n+1$ *times during the execution of the IMHA*.*

Proof: In IMHA*, a state s can only be expanded when it is selected as the top state of $OPEN_i$ in either line 22 or 27. In both the cases the very next call is to the function Expand, which removes this selected state from $OPEN_i$ (line 4). Now, a state (other than s_{start}) can only be inserted in $OPEN_i$ in line 11. If a state s has already been expanded in the i^{th} search, the check at line 10 will ensure that s is not inserted again in $OPEN_i$, and therefore cannot be expanded in the i^{th} search any more. In other words, a state s can only be expanded *at most* once in every search, and the total number of searches is $n+1$ (*anchor* search and n inadmissible searches). ■

Theorem 5. *In IMHA*, a state* s *is never expanded in the* i^{th} *inadmissible search if* $key(s,i) > w_1 * w_2 * g^*(s_{goal})$.

Proof: This theorem can be proved using Theorem 2, which states that $OPEN_0.Minkey() \leq w_1 * g^*(s_{goal})$. If a state s is selected for expansion in the i^{th} inadmissible search at line 22, it has $key(s,i) \leq OPEN_i.Minkey()$ (from the priority queue properties). Now, from the check at line 19, we obtain $OPEN_i.Minkey() \leq w_2 * OPEN_0.Minkey()$, otherwise the i^{th} search will be suspended and the control will not reach line 22. Therefore, if a state s is expanded in the i^{th} search, we have

$$\begin{aligned} key(s,i) &\leq OPEN_i.Minkey() \\ &\leq w_2 * OPEN_0.Minkey() \\ &\leq w_2 * w_1 * g^*(s_{goal}) \end{aligned} \tag{5}$$

■

Theorem 3 guarantees the suboptimality bounds for IMHA* while Theorems 4 and 5 provide the bounds on state expansions by IMHA*. The efficiency of IMHA* stems from the fact that it terminates as soon as any one of the n heuristics leads the search to a solution within the suboptimality bound, i.e., the total state expansions $\approx n \times$ minimum of the state expansions among all the searches. Thus, if any of the inadmissible searches converges faster than the *anchor* search by a factor better than n, IMHA* can outperform WA*.

B. Shared Multi-Heuristic A (SMHA*)*

```
1  procedure key$(s,i)$
2  return $g(s) + w_1 * h_i(s)$;
3  procedure Expand$(s)$
4  Remove $s$ from $OPEN_i$ $\forall i = 0..n$;
5  for each $s'$ in Succ$(s)$
6   if $s'$ was never visited
7    $g(s') = \infty$; $bp(s') = $ null;
8   if $g(s') > g(s) + c(s,s')$
9    $g(s') = g(s) + c(s,s')$; $bp(s') = s$;
10    if $s'$ has not been expanded in the anchor search
11     insert/update $(s')$ in $OPEN_0$ with $key(s',0)$;
12     if $s'$ has not been expanded in any inadmissible search
13      for $i = 1$ to $n$
14       if $key(s',i) \leq w_2 * key(s',0)$
15        insert/update $(s')$ in $OPEN_i$ with $key(s',i)$;
16 procedure SMHA*()
17 $g(s_{goal}) = \infty$; $bp(s_{start}) = bp(s_{goal}) = $ null;
18 $g(s_{start}) = 0$;
19 for $i = 0$ to $n$
20  $OPEN_i = \emptyset$;
21  insert $s_{start}$ into $OPEN_i$ with $key(s_{start},i)$ as priority;
22 while $OPEN_0$ not empty
23  for $i = 1$ to $n$
24   if $OPEN_i.Minkey() \leq w_2 * OPEN_0.Minkey()$
25    if $g(s_{goal}) \leq OPEN_i.Minkey()$
26     terminate and return path pointed by $bp(s_{goal})$;
27    $s = OPEN_i.Top()$;
28    Expand$(s)$;
29   else
30    if $g(s_{goal}) \leq OPEN_0.Minkey()$
31     terminate and return path pointed by $bp(s_{goal})$;
32    $s = OPEN_0.Top()$;
33    Expand$(s)$;
```

Fig. 3: Shared Multi-Heuristic A* (SMHA*)

The primary difference between SMHA* and IMHA* is that in SMHA*, the current path for a given state is shared among all the searches, i.e., if a better path to a state is discovered by any of the searches, the information is updated in all the priority queues. As the paths are shared, SMHA* uses a single g (and bp) value for each state, unlike IMHA* in which every search maintains its own g value. Furthermore, path sharing allows SMHA* to expand each state at most twice in contrast to IMHA* which may expand a state up to $n+1$ times (once in each search). We include the pseudocode for SMHA* in Figure 3.

The key function and initialization part in SMHA* is same as in IMHA* other than the fact that SMHA* uses a single g (and bp) variable. After the initialization, SMHA* runs the inadmissible searches in a round robin manner as long as the check in line 24 is satisfied. If the check is violated for a given search, it is suspended and a state is expanded from $OPEN_0$. The key difference between SMHA* and IMHA* lies in the state expansion method (Expand routine). In SMHA*, when a state s is expanded, its children ($s' \in Succ(s)$) are simultaneously updated in all the priority queues, if s' has not yet been expanded (lines 13-15). If s' has been expanded in any of the inadmissible searches but not in the *anchor* search (check at line 10), it is inserted only in $OPEN_0$. A state s' that has been expanded in the *anchor* search is never reexpanded and thus, never put back to any of the priority queues. The only exception to this simultaneous update (for a state s' not yet expanded) is the optimization at line 14 which ensures that s' is not put into $OPEN_i$ if $key(s',i) > w_2 * key(s',0)$, because such a state will never be expanded from $OPEN_i$ anyway (check at line 24). The Expand routine also removes s from all $OPEN_i$ (line 4) making sure that it is never reexpanded again in any inadmissible search and not reexpanded in the *anchor* search if its g is not lowered. If $g(s_{goal})$ becomes the minimum key in any of the searches, SMHA* terminates with a solution within $w_1 * w_2$ bound, otherwise no finite cost solution exists.

1) SMHA Properties:* While discussing the analytical properties for SMHA*, we should note that unlike IMHA*, the *anchor* search in SMHA* is not a direct replica of a single shot WA* with a consistent heuristic function. Thus, we cannot directly use the results for WA* (Theorem 1 and 2) to derive SMHA* properties. However, the next two theorems show that although the *anchor* search here is different, it essentially follows the same lower bound properties proved for IMHA*.

Theorem 6. *At line 23, for any state s with $key(s, 0) \leq key(u, 0) \forall u \in OPEN_0$, it holds that $g(s) \leq w_1 * g^*(s)$.*

Theorem 7. *At line 23, for any state s with $key(s, 0) \leq key(u, 0) \forall u \in OPEN_0$, it holds that $key(s, 0) \leq w_1 * g^*(s_{goal})$.*

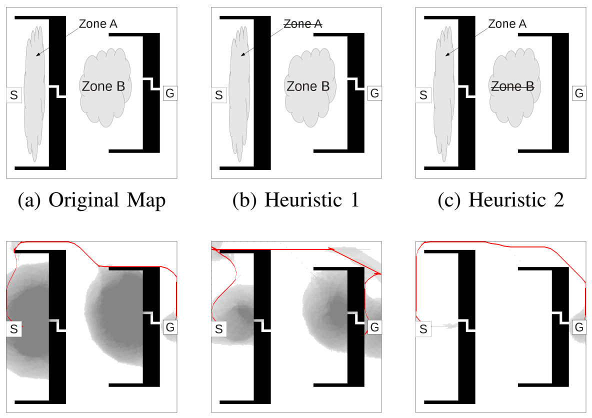
The complete proofs for these theorems are included in [3]. At an intuitive level, we can see that the lower bound results will be equivalent for IMHA* and SMHA* *anchor* searches, as at a given point the $OPEN_0$ in SMHA* can be seen as a superset of $OPEN_0$ in IMHA* (or WA* without reexpansions). This is due to the fact that whenever a state s is expanded in any of the searches of SMHA* its children are put into $OPEN_0$, thus it includes states from different searches. On the other hand, although s is deleted from $OPEN_0$ at this point (line 4), it can be re-inserted later if a better path to it is discovered (lowered g value), as long as it is not expanded in the *anchor* search. Now, as both Theorems 6 and 7 relate to the minimum key values in $OPEN_0$ and minimum of a superset is always less than or equal to the minimum of a subset, these results are valid for the *anchor* search in SMHA*. Next, we present three theorems characterizing the properties of SMHA*.

Theorem 8. *When SMHA* exits, $g(s_{goal}) \leq w_1 * w_2 * g^*(s_{goal})$, i.e., the solution cost is bounded by $w_1 * w_2$ suboptimality factor.*

Proof: This theorem can be proved in a manner similar to the proof for Theorem 3 using Theorems 6 and 7. ■

Theorem 9. *During the execution of SMHA*, a) no state is expanded more than twice, b) a state expanded in the anchor search is never reexpanded, and c) a state expanded in an inadmissible search can only be reexpanded in the anchor search if its g value is lowered.*

Proof: In SMHA*, a state s can only be expanded when it is selected as the top state of $OPEN_i$ in either line 27 or 32. If s is selected for expansion in line 27, the very next call is to the function Expand (line 28), which removes this selected state from $OPEN_i, \forall i = 0..n$ (line 4). Now, a state (other than s_{start}) can only be inserted in $OPEN_i$ ($i \neq 0$) in line 15. If a state s has already been expanded in any of the inadmissible searches, the check at line 12 will ensure that s is not inserted again in $OPEN_i$ ($i \neq 0$).Therefore, a state can only be expanded once in the inadmissible searches. Now, when a state s is expanded in the *anchor* search, similar to the earlier case, here also, s is removed from all $OPEN_i$ (line 4). Thus, s can only be expanded again either in inadmissible searches or in *anchor* search, if it is re-inserted in any of the $OPEN_i$, which can only be done in lines 11 or 15. However, as s has been expanded in the *anchor* search, the check at line 10 will never be true, thus the control will never reach lines 11 or 15, i.e., s will never be reexpanded. Therefore, statement b) is true. Also, as s can be expanded at most once in the *anchor* search and at most once in the inadmissible searches, s cannot be expanded more than twice, proving statement a). Finally, a state s that has been expanded in an inadmissible search, can only be expanded in the *anchor* search later if it is re-inserted in $OPEN_0$. A state can only be inserted in $OPEN_0$ (any $OPEN_i$, for that matter) if the check at line 8 is true, i.e., if its g value is less than its earlier g value. Thus, a state s whose g has not been lowered after its expansion in any inadmissible search will never satisfy the condition at line 8 and will not be re-inserted in $OPEN_0$ and thus, can never be expanded in the *anchor* search. Therefore, statement c) is true. ■



(a) Original Map (b) Heuristic 1 (c) Heuristic 2

(d) WA*(10.0) (e) IMHA*(10.0) (f) SMHA*(10.0)

Fig. 4: A 3D planning example with nested depression zones. The original map is shown in 4a. We compute two additional heuristics by blocking one narrow passage in the map as shown in 4b and 4c (using the PH procedure described in Section IV-B). As each inadmissible heuristic leads the search to a separate depression zone, IMHA* can not avoid any of them. However, SMHA* (4f) efficiently avoids both depression regions by using the first heuristic to go around zone A and the second heuristic to go around zone B and thus performs much better than WA* (4d) and IMHA* (4e).

Theorem 10. *In SMHA*, a state s is never expanded in the i^{th} inadmissible search if $key(s, i) > w_1 * w_2 * g^*(s_{goal})$.*

Proof: The proof is similar to Theorem 5, utilizing the fact that $OPEN_0.Minkey() \leq w_1 * g^*(s_{goal})$ (Theorem 7). ■

Theorem 8 shows that SMHA* guarantees the same sub-optimality bounds as IMHA* while Theorem 9 highlights the difference in complexity between these two approaches. In IMHA*, a state can be reexpanded at most $n + 1$ times as each search is performed independently, whereas in SMHA* the same bounds are attained with at most 1 reexpansion per state. A more important distinction between SMHA* and IMHA* arises from the fact that as SMHA* shares the best path information among all the searches, it can potentially use a combination of partial paths to exit from depression regions, which is not possible in IMHA*. Therefore, if there are nested depression regions in the state space that none of the inadmissible heuristics can avoid independently, SMHA* can outperform IMHA*. In Figure 4, we illustrate this phenomenon with an example of a search space with nested depression zones. On the other hand, IMHA* has the following advantages over SMHA*, a) expansion of states in IMHA* is cheaper than SMHA* as SMHA* may require $n+1$ insertion/update/removal steps, b) in SMHA* all the searches store a copy of each of the generated states, thus the memory overhead is more[1], and c) as IMHA* does not share paths, it is more amenable to parallelization.

IV. Experimental Results

We evaluated the performance of the MHA* algorithms for the following domains: 12D mobile manipulation planning for the PR2 robot, 3D (x, y, orientation) navigation, and sliding tile puzzles. We primarily benchmarked MHA*s against WA* without reexpansions (as in ARA* [19]). We also compared with the sampling based planning algorithms (PRM [16], RRT-Connect [14], and RRT* [15]), multiple heuristic greedy best first search (MHGBFS) [25], multiple parameter WA* (MPWA*) [29] and Explicit Estimation Search (EES) [27] when applicable. For MHGBFS, we used the same heuristics as used for SMHA*. For MPWA*, we used the admissible heuristic with 5 different weights ($0.2 \times w$ to $1.0 \times w$, with 0.2 gap; where $w \geq 10.0$). For EES, we used an inadmissible distance measure, one inadmissible heuristic function (from the set used for MHA*s) and the admissible heuristic. We ran WA*, MPWA* and MHGBFS without state reexpansions as reexpansions can degrade the planning time. WA* and MPWA* can satisfy the quality bounds without reexpansions, while MHGBFS does not guarantee any bounds. We performed all the experiments on an Intel $i7-3770$ ($3.40GHz$) PC with $16GB$ RAM. As MHA*s use two suboptimality bounds (w_1 and w_2) in comparison to one w used by WA*/MPWA*/EES, we set $w_2 = min(2.0, \sqrt{w})$ and $w_1 = w/w_2$, for all our experiments (and the examples), so that the solutions are guaranteed to be within w-suboptimality bound.

A. Mobile Manipulation Planning for the PR2 (12D)

The PR2 mobile manipulation robot is a dual-arm robot (7 degrees of freedom each) with an omni-directional base and a prismatic spine. In our experiments, we used a state space representation similar to Cohen et al. [8]. We represented a robot state with 12 degrees of freedom: a 6 DOF object pose, 2 redundant arm joints, 2D Cartesian coordinates for the base, an orientation of the base, and the prismatic spine height. The planner was provided the initial configuration of the robot as the start state. The goal state contained only the 6 DOF position of the object, which made it inherently under-specified because it provided no constraints on the position of the robot base or the redundant joint angles. The actions used to generate successors for states were a set of motion primitives, which are small, kinematically feasible motion sequences that move the object in 3D Cartesian space, rotate the redundant joint, or move the base in a typical lattice-type manner [20]. The prismatic spine was also allowed to adjust its height in small increments.

[1]It may be noted that this memory overhead can be eliminated by using a single open list and making the update/insertion/removal more informed.

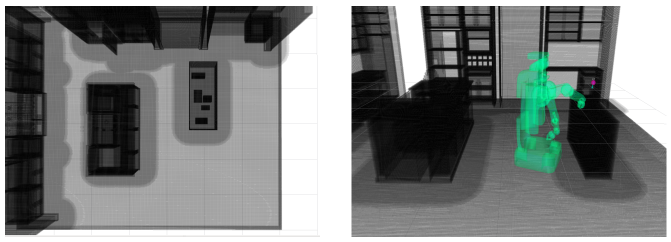

(a) (b)

Fig. 5: Example of kitchen environments used for mobile manipulation planning.

We computed the admissible heuristic by taking the maximum value between the end-effector heuristic and the base heuristic described next. The end-effector heuristic was obtained by a 3D Dijkstra search initialized with the (x,y,z) coordinates of the goal and with all workspace obstacles inflated by their inner radius. The base heuristic was obtained using a 2D Dijkstra search for the robot base where the goal region is defined by a circular region centered around the (x,y) location of the 6 DOF goal. The purpose of this circular region is to maintain an admissible heuristic despite having an incomplete 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.

	WA*	MHGBFS	MPWA*	EES	IMHA*	SMHA*
SR	31%	76%	36%	27%	70%	81%
SE	1.08	0.78	3.84	1.54	1.58	1.0
RT	0.99	0.91	2.82	1.54	1.41	1.0
SC	0.95	1.57	0.97	0.93	1.09	1.0

TABLE I: Comparison between WA*, MHGBFS, MPWA*, EES and MHA*s for PR2 manipulation planning in kitchen environments. The first row (SR) shows the percentage of total problem instances solved by each planner. The other rows include the results as a ratio between the algorithm marked in the column heading and the corresponding SMHA* numbers, when both of them solved an instance. Legend: SR - Success Rate, SE - state expansion ratio, RT - runtime ratio, SC - solution cost ratio.

For IMHA*, we computed 2 additional heuristics in the following way. We randomly selected 2 points (with valid IK solutions for the arm to reach the goal) from the base circle around the goal and ran 2D Dijkstra searches starting from these 2 points to compute the inadmissible base distances. We also computed an inadmissible orientation distance by obtaining the Euclidean distance between the current base orientation (at a given point) and the desired orientation, which was to make the robot face the end-effector goal. These inadmissible distances (base and orientation) were then added to the end-effector to compute the final heuristic values. Note that this informative heuristic is clearly inadmissible, but can still be used in the MHA* framework. For SMHA*, we augment this set by using the base (2D Dijkstra + orientation) and the end-effector heuristics (3D Dijkstra) as two additional heuristics, since SMHA* can share the paths among the inadmissible searches, and hence, can potentially benefit from not combining the two heuristics into a single one. The test environment for this experiment was a kitchen-like environment with randomly generated obstacles. In Figure 5a (top view) and 5b (with robot and goal position), we include an example of the test scenario with two large tables and a few narrow passageways. For each trial of the experiment, we randomly generated a full robot configuration anywhere in the kitchen for the start state, while generating a valid goal state that lies above the tabletops containing clutter. We generated 15 such environments by randomly changing the object positions and for each such environment we used 10 different start and goal configurations.

In Table I, we include the results comparing WA*, EES, MPWA*, MHGBFS with the MHA*s. We used $w = 50$ for all the algorithms. Each planner was given a maximum of 60 seconds to compute a plan. The results clearly show that MHA*s (especially SMHA*) and MHGBFS perform much better than WA*/MPWA*/EES, highlighting the efficacy of using multiple heuristics over a single heuristic function, which often suffers from local minima due to robots orientation, presence of obstacles, etc. MPWA* performs slightly better than WA* indicating that the size of a local minimum can depend on the weights used, however it still gets stuck in most of the cases since it uses the same heuristic (albeit with different weights) for each search. EES performs poorly when the inadmissible distance function has a large depression. Also, the inadmissible and admissible searches in EES do not use weighted heuristics and thus, often get trapped in some cost plateau. MHA*s (and MHGBFS) are less prone to suffer from heuristic depression regions as they can converge in time if any of the heuristics can lead the search to the goal. SMHA* and MHGBFS perform better than IMHA*, as they can use partial paths. For example, they can combine a path obtained in the base heuristic search with the end-effector heuristic search. MHGBFS performs comparably to SMHA* in terms of number of instances solved and slightly better in terms of convergence time. However, the solution costs obtained for MHGBFS are significantly worse than SMHA* (and IMHA*), as noted in Solution Cost ratio in Table I. This highlights the utility of the *anchor* search, which ensures better quality solution by intelligently controlling the inadmissible expansions.

	PRM	RRT-Connect	RRT*(First)	RRT*(Final)	IMHA*	SMHA*
SR	74%	98%	100%	100%	70%	81%
RT	2.07	0.18	5.39	8.48	1.41	1.00
BD	1.93	1.88	1.36	1.34	1.02	1.00
ED	1.87	1.68	1.27	1.24	0.99	1.00

TABLE II: Comparison between MHA*s and sampling based planners for PR2 manipulation in kitchen environments. All the results are presented as a ratio between the algorithm marked in the column heading and the corresponding SMHA* numbers. For sampling based planners, the distances are obtained after post processing. Since RRT* is an anytime algorithm, we include the results for the first solution reported (RRT*-First) and the solution obtained at the end of 60 secs (RRT*-Final). Legend: SR - Success Rate, RT - runtime ratio, BD - base distance ratio, ED - end effector distance ratio.

In Table II, we include the results comparing MHA*s with 3 sampling based algorithms, namely PRM, RRT-Connect and RRT*, in terms of runtime and solution quality. For the sampling based algorithms we used the standard OMPL [26] implementation. Since the sampling-based planners do not directly report the planning costs, in this table we include the results in terms of base and end-effector distances covered by the robots (after post processing). All the results are presented as a ratio over the corresponding SMHA* numbers (for episodes where both were successful). The results show that SMHA* performs reasonably well when compared to the sampling based planners, runtime-wise it is better than both PRM and RRT* but worse than RRT-Connect. In terms of solution quality, MHA*s are noticeably better than all the sampling based planners. However, both RRT-Connect and RRT* can solve more number of instances, mainly due to the facts that a) they are not bound by discretization choices and b) they do not use any heuristic function that may lead to local minima.

B. 3D Path Planning (Navigation)

While high dimensional problems like full-body planning for the PR2 are a true testbed for assessing the real life applicability of MHA*s, finding close-to-optimal solutions in such spaces is infeasible. Therefore, in order to get a better idea of MHA*s' behavior for close-to-optimal bounds, we ran experiments in an easier 3D (x, y, orientation) navigation domain. The search objective here is to plan smooth paths that satisfy the constraints on the minimum turning radius. We used two kinds of maps (1000×1000 dimensions): i) indoor maps, that are composed of randomly placed narrow hallways and large rooms with polygonal obstacles, and ii) outdoor maps, that have large open spaces with random regular obstacles.We computed the consistent heuristics (h_0) by running a 16-connected 2D Dijkstra search by inflating the objects using the robot's (PR2 base) in-radius. For generating the additional heuristics, we used two strategies, i) dual heuristics: where we generated an extra heuristic by performing another 2D Dijkstra search by inflating the obstacles using the robot's out-radius, and ii) progressive heuristics: where the 2D Dijkstra path obtained for h_0 is investigated for possible bottlenecks, if the path has narrow passages, those are blocked and a new heuristic is computed. We used two progressive schemes, PH where the above mentioned procedure is iterated until a *relatively wide* 2D path is discovered and PG where the PH heuristic set is augmented with an extra heuristic computed by using a tunnel around the last 2D path.

	Indoor Environments				Outdoor Environments			
	IMHA*		SMHA*		IMHA*		SMHA*	
w	RT	SC	RT	SC	RT	SC	RT	SC
10.0	0.25	0.88	0.28	0.87	1.22	0.78	1.14	0.76
5.0	0.27	0.97	0.27	1.00	1.05	0.87	1.03	0.86
3.0	0.32	1.00	0.34	1.00	0.97	0.87	0.71	0.87
2.0	0.39	1.01	0.36	0.97	0.59	0.89	0.40	0.90
1.5	1.18	1.01	0.84	0.92	0.68	0.91	0.39	0.92

TABLE III: Comparison between WA* and MHA*s with dual heuristics for indoor and outdoor maps. All the results are shown as a ratio between the algorithm marked in the column heading and the corresponding WA* numbers. Legend: RT - runtime ratio, SC - solution cost ratio.

In Tables III and IV, we include the results comparing MHA*s with WA* for dual heuristics and progressive heuristics, respectively. For indoor maps, the PH scheme generated additional heuristics for 44 maps (out of 100), for outdoor maps, it generated additional heuristics for 8 maps only.

	Indoor Environments								Outdoor Environments							
	IMHA*		SMHA*		IMHA*		SMHA*		IMHA*		SMHA*		IMHA*		SMHA*	
	PH		PH		PG		PG		PH		PH		PG		PG	
w	RT	SC	RT	SC	RT	SC	RT	SC	RT	SC	RT	SC	RT	SC	RT	SC
10.0	0.16	0.81	0.13	0.87	0.14	0.73	0.05	0.73	1.46	0.83	1.38	0.75	1.38	0.79	1.38	0.55
5.0	0.18	0.79	0.13	0.88	0.15	0.64	0.07	0.64	1.50	0.81	1.22	0.73	1.17	0.85	1.06	0.59
3.0	0.19	0.93	0.21	0.88	0.24	0.93	0.12	0.81	1.07	0.92	1.04	0.80	1.21	0.93	0.82	0.81
2.0	0.82	0.88	0.40	0.87	0.83	0.91	0.19	0.89	1.06	0.97	0.96	0.80	1.09	0.91	0.74	0.88
1.5	1.15	0.92	0.79	0.83	1.13	0.91	0.52	0.89	1.20	0.97	0.89	0.94	1.16	0.96	0.87	0.94

TABLE IV: Comparison between WA* and MHA*s with PH and PG heuristics. Runtime numbers include both planning and heuristic generation times. Legend: RT - runtime ratio, SC - solution cost ratio.

The results show that for indoor maps, MHA*s generally outperform WA* by a significant margin, whereas for outdoor maps, all the algorithms perform similarly (MHA*s are a trifle worse). This is because, in indoor maps, the presence of large rooms and narrow corridors frequently creates big depression regions. MHA*s can utilize the additional inadmissible heuristics to quickly get away from such depression zones, but WA* cannot. In contrast, the outdoor maps are generally more benign and large depression regions are rare, thus, WA* is very efficient. However, even for outdoor environments, MHA*s (especially SMHA*) perform better in case of low bounds. The performance gap is more pronounced with the PH and PG schemes as the number of heuristics increases.

	MHGBFS	MPWA*	IMHA*	SMHA*
IS	37	35	52	52
RT	2.39	3.38	1.35	1.00
SC	7.27	1.26	1.22	1.00

TABLE V: Comparison between MHGBFS, MPWA*, IMHA* and SMHA* for 3D path planning (As a ratio over SMHA* results). All the planners were given maximum 5 secs to plan. Legend: IS - number of instances solved (out of 52), RT - runtime ratio, SC - solution cost ratio.

Table V includes the results comparing MHA*s (PG) with the MHGBFS and MPWA* on the combined set of 52 *hard* instances (44 indoor + 8 outdoor), for which the PH scheme generated extra heuristics. For MHA*s and MPWA*, we used $w = 10.0$. Each algorithm was given a maximum of 5 seconds to compute a plan. Comparison between MPWA* and MHA*s show a similar trend as seen for the 12D planner. MHGBFS performance degrades considerably for this domain as its greedy approach at times drives the search deeper into a local minimum. Unlike 12D planning, cost plateaus are rare here and thus, the greedy approach hurts more often than it helps.

C. Sliding Tile Puzzle

For sliding tile puzzles, we used the Manhattan distance (MD) plus linear conflicts (LC) [1] as the consistent heuristic (h_0). For MHA*s, we generated 4 additional heuristics by computing the number of misplaced tiles (MT), and adding MD, LC and MT with random weights between 1.0 and 5.0, i.e., we used $h_i = r_1 * MD + r_2 * LC + r_3 * MT$, where r_1, r_2, r_3 are random numbers ≥ 1.0 and ≤ 5.0. Clearly, these heuristics are somewhat random and therefore, easy to design. We tested the algorithms on 50 random instances (all solvable) of 48, 63 and 80 puzzles.

In Table VI, we include the results in terms of the number of problems solved by all the algorithms under two time limits, 1 minute and 3 minutes. The results show that even with such arbitrarily computed heuristics, MHA*s, especially

Size	Bound	WA*	IMHA*	SMHA*	WA*(R)	WA*	IMHA*	SMHA*	WA*(R)
48	50	46	49	50	49	47	49	50	50
	20	49	47	50	43	49	49	50	46
	10	45	37	50	32	46	45	50	32
	5	37	19	49	-	38	39	50	-
	2	12	4	9	-	12	6	10	-
63	50	25	35	40	26	29	38	44	33
	20	34	26	39	18	35	37	41	31
	10	32	21	39	17	35	29	40	18
	5	19	8	31	-	24	19	37	-
	2	3	4	4	-	7	4	9	-
80	50	17	24	31	15	22	26	33	23
	20	22	17	27	13	22	27	37	16
	10	19	19	29	12	21	25	30	18
	5	17	11	22	-	20	14	28	-
	2	7	1	4	-	7	1	9	-

TABLE VI: Number of sliding tile puzzle instances solved by WA*, IMHA* and SMHA* for different suboptimality bounds with time limit 1 minute (columns 3, 4 and 5) and 3 minutes (columns 7, 8 and 9). WA*(R) columns (6 and 10) show the results obtained by WA* using the same randomized heuristic as used for MHA*.

SMHA*, can outperform WA*. The performance gap is more pronounced for larger sized problems and higher w values. We also include the results for WA* with the same heuristic function $h_r = r_1 * MD + r_2 * LC + r_3 * MT$ (referred to as WA*(R)). As $h_r \leq 10 * h_0$, we computed the results for suboptimality bounds ≥ 10 only. From Table VI, we see that the WA*(R) results are actually worse, indicating that MHA*s' improved performance is due to their multi-heuristic exploration.

Size		MHGBFS	MPWA*	IMHA*	SMHA*
48	IS	50	44	38	50
	RT	0.63	2.57	2.63	1.00
	SC	3.59	1.02	0.96	1.00
63	IS	29	27	26	39
	RT	1.06	1.46	1.13	1.00
	SC	4.98	0.97	0.94	1.00
80	IS	21	16	17	27
	RT	0.85	1.17	0.91	1.00
	SC	3.92	0.94	0.99	1.00

TABLE VII: Comparison between MHGBFS, MPWA*, IMHA* and SMHA* for sliding tile puzzle. The maximum runtime allowed was 1 minute. Legend: IS - number of instances solved (out of 50), RT - runtime ratio, SC - solution cost ratio (over SMHA*).

In Table VII, we present the results obtained by comparing MHA*s to MHGBFS and MPWA*. MHA*s and MPWA* were run with $w = 20.0$. All the algorithms were given a time limit of 1 minute. From the results, we observe that for this domain, SMHA* consistently outperforms both MHGBFS/MPWA*. For each size, SMHA* solved more number of instances than both MPWA*/MHGBFS. Although MHGBFS had a better planning time in two scenarios, its solution quality was markedly worse, as one would expect from the greedy approach.

V. Conclusions and Future Work

We presented a heuristic search framework that uses multiple inadmissible heuristics to simultaneously explore the search space, while preserving guarantees of completeness and suboptimality bounds using a consistent heuristic. Experimental results obtained on various domains demonstrate the efficacy of the proposed framework, especially for search spaces with large depression regions. While the initial results with MHA*s are encouraging, we believe that there is scope for major improvements/extensions. Possible future extensions include anytime version of MHA*, dynamic recomputation of heuristics, and parallel MHA*.

References

[1] http://heuristicswiki.wikispaces.com/N+-+Puzzle.

[2] Sandip Aine, P. P. Chakrabarti, and Rajeev Kumar. AWA* - A Window Constrained Anytime Heuristic Search Algorithm. In Manuela M. Veloso, editor, *IJCAI*, pages 2250–2255, 2007.

[3] Sandip Aine, Venkatraman Narayanan, Siddharth Swaminathan, Victor Hwang, and Maxim Likhachev. MHA*: The Proofs. Technical Report TR-13-08, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2014.

[4] Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors. *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, 2010. AAAI.

[5] 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.

[6] P. P. Chakrabarti, Sujoy Ghose, and S. C. De Sarkar. Generalized best first search using single and multiple heuristics. *Inf. Sci.*, 60(1-2):145–175, 1992.

[7] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev. Single- and dual-arm motion planning with heuristic search. *International Journal of Robotics Research (IJRR)*, 2013.

[8] Benjamin J. Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for dual-arm manipulation with upright orientation constraints. In *ICRA*, pages 3784–3790. IEEE, 2012. ISBN 978-1-4673-1403-9.

[9] A. Felner, R. E. Korf, and S. Hanan. Additive Pattern Database Heuristics. *J. Artif. Intell. Res. (JAIR)*, 22:279–318, 2004.

[10] 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.

[11] C. Hernández and J. A. Baier. Avoiding and Escaping Depressions in Real-Time Heuristic Search. *J. Artif. Intell. Res. (JAIR)*, 43:523–570, 2012.

[12] Armin Hornung, Daniel Maier, and Maren Bennewitz. Search-based footstep planning. In *Proc. of the ICRA Workshop on Progress and Open Problems in Motion Planning and Navigation for Humanoids*, Karlsruhe, Germany, May 2013.

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

[14] James J. Kuffner Jr. and Steven M. LaValle. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *ICRA*, pages 995–1001. IEEE, 2000. ISBN 0-7803-5889-9.

[15] S. Karaman and E. Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. In *Robotics: Science and Systems*, Zaragoza, Spain, June 2010. The MIT Press.

[16] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces. *IEEE T. Robotics and Automation*, 12(4):566–580, 1996.

[17] R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22, 2002.

[18] Steven M. Lavalle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[19] 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.

[20] Maxim Likhachev and Dave Ferguson. Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles. *I. J. Robotic Res.*, 28(8):933–945, 2009.

[21] Brian MacAllister, Jonathan Butzke, Aleksandr Kushleyev, and Maxim Likhachev. Path Planning for Non-Circular Micro Aerial Vehicles in Constrained Environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3933–3940, 2013.

[22] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984. ISBN 0-201-05594-5.

[23] Judea Pearl and Jin H. Kim. Studies in Semi-Admissible Heuristics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4 (4):392–399, 1982.

[24] I. Pohl. Heuristic Search Viewed as Path Finding in a Graph. *Artif. Intell.*, 1(3):193–204, 1970.

[25] Gabriele Röger and Malte Helmert. The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning. In Brafman et al. [4], pages 246–249.

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

[27] Jordan Tyler Thayer and Wheeler Ruml. Bounded Suboptimal Search: A Direct Approach Using Inadmissible Estimates. In *IJCAI*, pages 674–679, 2011.

[28] Jordan Tyler Thayer, Roni Stern, Ariel Felner, and Wheeler Ruml. Faster Bounded-Cost Search Using Inadmissible Estimates. In Lee McCluskey, Brian Williams, José Reinaldo Silva, and Blai Bonet, editors, *ICAPS*. AAAI, 2012. ISBN 978-1-57735-562-5.

[29] Richard Anthony Valenzano, Nathan R. Sturtevant, Jonathan Schaeffer, Karen Buro, and Akihiro Kishimoto. Simultaneously Searching with Multiple Settings: An Alternative to Parameter Tuning for Suboptimal Single-Agent Search Algorithms. In Brafman et al. [4], pages 177–184.

[30] C. M. Wilt and W. Ruml. When Does Weighted A* Fail? In *SOCS*. AAAI Press, 2012.

[31] 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.

[32] Matt Zucker, Nathan Ratliff, Martin Stole, Joel Chestnutt, J. Andrew Bagnell, Christopher G. Atkeson, and James Kuffner. Optimization and learning for rough terrain legged locomotion. *International Journal of Robotics Research*, 30(2):175–191, 2011.