Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

# Speeding Up Search-Based Motion Planning via Conservative Heuristics

**Ishani Chatterjee, Maxim Likhachev, Ashwin Khadke, Manuela Veloso***
The Robotics Institute, Carnegie Mellon University

## Abstract

Weighted A* search (wA*) is a popular tool for robot motion-planning. Its efficiency however depends on the quality of heuristic function used. In fact, it has been shown that the correlation between the heuristic function and the true cost-to-goal significantly affects the efficiency of the search, when used with a large weight on the heuristics. Motivated by this observation, we investigate the problem of computing heuristics that explicitly aim to minimize the amount of search efforts in finding a feasible plan. The key observation we exploit is that while heuristics tries to guide the search along what looks like an optimal path towards the goal, there are other paths that are clearly sub-optimal yet are much easier to compute. For example, in motion planning domains like footstep-planning for humanoids, a heuristic that guides the search along a path away from obstacles is less likely to encounter local minima compared with the heuristics that guides the search along an optimal but close-to-obstacles path. We utilize this observation to define the concept of conservative heuristics and propose a simple algorithm for computing such a heuristic function. Experimental analysis on (1) humanoid footstep planning (simulation), (2) path planning for a UAV (simulation), and a real-world experiment in footstep-planning for a NAO robot shows the utility of the approach.

## Introduction

Weighted A* (wA*) (Pohl 1970), has been widely used for relatively low-dimensional motion planning problems such 2D navigation (Ferguson, Likhachev, and Stentz 2005), path planning for UAVs (Hwangbo, Kuffner, and Kanade 2007), (Liu et al. 2018), and footstep planning for humanoids (Hornung, Maier, and Bennewitz 2013). wA* with high weight shows better search efficiency in domains that show a strong correlation of the heuristic function with the node-distance-to-goal (Wilt and Ruml 2012). A weak correlation may create heuristic depression regions, or local minima, where the path suggested by the heuristic may not be feasible, severely degrading search efficiency (Wilt and Ruml 2012). We investigate the idea of computing heuristic functions that explicitly aim to reduce search expansions by wA*.

*This work was in part supported by ONR grant N00014-15-1-2129

For heuristic computation, it is common to solve a simpler planning problem in a space formed by relaxing some constraints in the original space (Bulitko et al. 2007), (Holte et al. 1996). We define an edge in the relaxed space as conservative if for each corresponding state in the original space there is an edge to at least one corresponding successor state. Existence of a path composed only of conservative edges in the relaxed space, guarantees existence of a feasible path in the original space. If the heuristic computation finds such a path in the relaxed space, then simply following the heuristic gradient can guide the search to the goal, while expanding only the states that appear in the solution. Our first contribution is in observing that in motion planning problems formulated as heuristic search one can often identify conservative edges in the relaxed space. Secondly, we propose a heuristic computation algorithm that minimizes the use of non-conservative edges to reduce inefficient expansions in the original space.

Motivation behind the conservative property is similar to several ideas explored in classical hierarchical planning. Relaxed spaces and "safe" abstractions (Haslum and others 2007) have been used to compute solutions that could be refined/extended to the original space (Bacchus and Yang 1991). Bacchus and Yang showed that plans in the relaxed space can be refined to a plan in original space with a high probability if the relaxed space satisfies the Downward Refinement Property (DRP). It is difficult to find non-trivial relaxed spaces with DRP. In our work, the relaxed spaces do not satisfy DRP, but we observe that we can identify conservative edges in the relaxed space that can direct the search in the original space towards goal. Also, we do not refine plans computed in the relaxed space but use them to compute a heuristic to be used by the original search. Many works use relaxed spaces to compute heuristics (Pearl 1984), (Hoffmann and Nebel 2001), (Holte et al. 1996). Pattern databases (Culberson and Schaeffer 1998) store a table mapping states or sub-goals in a relaxed version of the original problem, to the cost-to-go of a pre-computed solution in the relaxed space to reach these sub-goals. (Helmert et al. 2007) compute heuristics in abstract-spaces for automated-planning, where abstractions are computed using different sets of state-variables. The Fast Downward Planner attempts to combine refinement with heuristics (Helmert 2006). (Bäckström and Jonsson 2013) define the weak refinement property (WRP) in studying the relationship between refinement and heuristics, and (Pang and Holte 2011) define "strong matching", both of which come close to the conservative property, except that it is not a requirement in our relaxed spaces. Also, these works mainly deal with symbolic planning, whereas our focus is in motion planning. (Vega-Brown and Roy 2018) use abstraction-based search in motion-planning that divides the environment into overlapping regions and has the effect of heuristically guiding the search towards the next region, based on some computed bounds. However, they make assumptions about the convexity of the regions. In our case, there is no such assumption about the planning environment. To the best of our knowledge, no attempt has been made to identify and use conservative edges for heuristic computation in the context of motion planning.

## Planning with Conservative Heuristics

### Definitions, Notations and Problem Description

Consider a graph $G = (S, E, c)$, where $S$ is the set of states, $E = \{(s, s')|s, s' \in S\}$ denotes the set of feasible transitions/edges in the graph and, $c$ is a cost-function such that $c(s_i, s_j)$ is the cost of an edge $(s_i, s_j)$. A planning problem consists of finding a path $\pi(s_i, s_j)$ in $G$ from $s_i$ to $s_j$. $\pi^*(s_i, s_j)$ denotes the least-cost path between $s_i$ and $s_j$. The cost of any path $\pi(s_i, s_j)$ is the cumulative cost of all edges along it and is denoted by $c(\pi(s_i, s_j))$. Let $h_c : S \rightarrow \mathbb{N}$ be our conservative heuristic function estimating cost-to-goal. We assume that $s_i \in S$ is a goal-state if and only if $h_c(s_i) = 0$. We use wA* to compute a path in G from a start state $s_{st}$ to any state in the goal-set $S_g = \{s \in S \mid h_c(s) = 0\}$.

**Heuristic Space:** Consider another state-space $\tilde{S}$, an abstract space used to compute heuristics. We call it the heuristic-space. Let $\lambda : S \rightarrow \tilde{S}$ be a many-to-one mapping representing the projection of each state in $S$ to the heuristic-space $\tilde{S}$, such that $|\tilde{S}| < |S|$. Moreover, $\lambda^{-1}(\tilde{s}) = \{s \in S \mid \lambda(s) = \tilde{s}\}\ \forall \tilde{s} \in \tilde{S}$. The heuristic-space has its own set of transitions $\tilde{E} = \{(\tilde{s_i}, \tilde{s_j})|\tilde{s_i}, \tilde{s_j} \in \tilde{S}\}$. Let $\tilde{G}$ be the graph defined by $\tilde{S}$ and $\tilde{E}$. $\pi(\tilde{s}_i, \tilde{s}_j)$ denotes a path in $\tilde{G}$ from $\tilde{s}_i$ to $\tilde{s}_j$, and $c(\pi(\tilde{s}_i, \tilde{s}_j))$ denotes its cost in $\tilde{G}$. We assume that for every pair of states $s_i$ and $s_j$ in $S$,

$$c(\pi^*(s_i, s_j))) \geq c(\pi^*(\lambda(s_i), \lambda(s_j))) \qquad (1)$$

$$\begin{aligned} &\text{if } \exists\ \pi(s_i, s_j)\ \ s.t\ \ |c(\pi(s_i, s_j))| < \infty, \\ &\text{then } \exists\ \pi(\lambda(s_i), \lambda(s_j))\ \ s.t\ \ |c(\pi(\lambda(s_i), \lambda(s_j)))| < \infty \end{aligned} \qquad (2)$$

We assume states in the goal-set $S_g$ map to one goal-state $\tilde{s}_g$ in the heuristic space, ie, $\tilde{s}_g = \lambda(s_g) \forall s_g \in S_g$.

**Conservative Edges:** An edge $(\tilde{s}, \tilde{s}') \in \tilde{E}$ is conservative iff $\forall s \in \lambda^{-1}(\tilde{s})\ \exists\ (s, s') \in E$ s.t $s' \in \lambda^{-1}(\tilde{s}')$. A conservative edge $(\tilde{s}, \tilde{s}')$ guarantees that every state $s \in \lambda^{-1}(\tilde{s})$ is connected to at least one state in $\lambda^{-1}(\tilde{s}')$. Thus, existence of a path in the heuristic-space from $\lambda(s_i)$ to $\lambda(s_j)$ which consists of only conservative edges, guarantees existence of a path from $s_i$ to $s_j$ in the original space. Fig. 1 illustrates this reasoning. If the search in $G$ is guided to always prefer successors connected via conservative edges in $\tilde{G}$, it would reach the goal by expanding only such successors, making the number of expansions equal to the solution size.

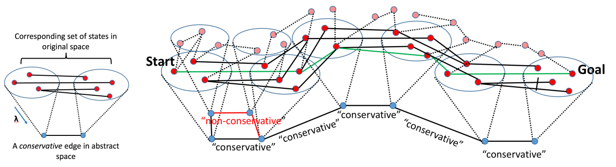

Figure 1: (a) A conservative edge in ($\tilde{S}$) and its corresponding set of states and edges in ($G$). (b) Non-conservative edges (red) with missing edges between some pairs of corresponding states in $S$. (c) Path (green) in $S$, comprising solely of conservative edges in $\tilde{S}$.

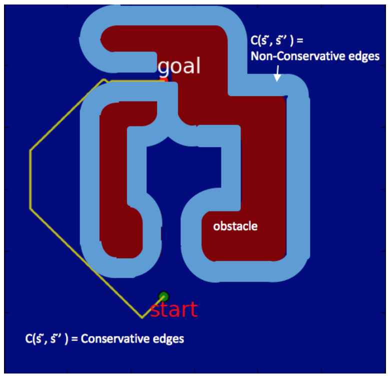
goal
C(s̃, s̃') = Non-Conservative edges
obstacle
start
C(s̃, s̃') = Conservative edges

Figure 2: A planning environment with obstacles (red) inflated by the circumradius of a polygonal robot. Edges inside the inflated region (light blue) are non-conservative, while those outside (dark blue) are conservative.

Identifying conservative edges in $\tilde{G}$ is a domain-dependent process. For example, when planning in $S = (x, y, \mathit{orientation})$ for a non-circular robot, $\tilde{S}$ can be obtained by dropping the orientation, or, $\tilde{S} = (x, y)$. Here, the heuristic-space is obtained by abstracting away a specific geometric property of the robot (in this case the orientation), which is equivalent to treating the non-circular robot as a circular one in the heuristic-space. Inflating obstacles by the radius of the robot-footprint's circumcircle (circumradius) identifies the space that the robot can physically occupy for *all* possible orientations. States in $\tilde{S}$ lying outside of this inflated-obstacle region are surely not in collision with obstacles, therefore an omni-directional robot can surely move between neighbouring states. Thus, edges between these states are conservative. Fig. 2 shows the conservative and non-conservative edges in a 2D environment inflated by the robot circumradius.

### Conservative Heuristic Computation

We want to compute $h_c(s)$ such that it (1) guides the search in the original space along paths that minimize the number of non-conservative edges in $\tilde{S}$, (2) prefers the shortest between all paths made purely of conservative edges and, (3) is $\alpha$ consistent ($\infty > \alpha > 1$). We define a heuristic to be $\alpha$-consistent if for all $s, s' \in S$ such that $s'$ is a successor of $s$, $h_c(s) \leq \alpha c(s, s') + h_c(s')$, and $h_c(s) = 0$ for all $s \in S_g$. Consider a graph $\tilde{G}_m = (\tilde{S}, \tilde{E}, c_m)$, which is $\tilde{G}$ but with modified edge-cost. For any $s \in S$ and its image $\tilde{s} = \lambda(s)$, we define $h_c(s) = c_m(\pi^\star_m(\tilde{s}, \tilde{s}_g))$, where $\pi^\star_m$ is the optimal path in $\tilde{G}_m$ from $\tilde{s}$ to $\tilde{s}_g$, Let $\tilde{E}_{co}$ be the set of conservative edges in $\tilde{E}$, $\tilde{E}_{nco} = \tilde{E} \setminus \tilde{E}_{co}$ . Let $c_{min}$ be the minimum edge-cost in $\tilde{G}$. We define $c_m$ in Eq. (3).

$$c_m(\tilde{s}_i, \tilde{s_j}) = \begin{cases} c_{min}/|\tilde{E}_{co}| & \text{if } (\tilde{s}_i, \tilde{s_j}) \in \tilde{E}_{co} \\ \alpha c_{min} & \text{if } (\tilde{s}_i, \tilde{s_j}) \in \tilde{E}_{nco} \end{cases} \tag{3}$$

From Eq. (3), we see that the optimal path ($\pi^\star_m(\tilde{s}, \tilde{s}_g)$) in $\tilde{G}_m$ would rather consist of all possible conservative edges than incorporating a single non-conservative one. Thus, the desired properties of $h_c$ are achieved.

Algorithm 1 shows the heuristic computation in detail. We need to first compute $c_{min}$ and total number of conservative edges $|\tilde{E}_{co}|$ (Lines 1:5). We then compute the shortest path in $\tilde{G}_m$ from $\tilde{s}_g$ to every state in $\tilde{G}_m$. This is done by running a backward Dijstra's search on $\tilde{G}_m$ from $\tilde{s}_g$ to every state in $\tilde{G}_m$. $\tilde{G}_m$ is implicitly constructed: for each expanded $\tilde{s}$ and a predecessor $\tilde{s}'$, we check whether $(\tilde{s}, \tilde{s}')$ is conservative and assign costs according to the scheme described in Eq. (3) (Lines 6:19).

For each state $s$ expanded by wA* search in the original graph $G$, we first find its projection $\tilde{s} = \lambda(s)$. $g(\tilde{s})$ which was updated when $\tilde{s}$ was expanded in the heuristic computation search, is the cost of the shortest path in $\tilde{G}_m$. Therefore, $h_c(s) = g(\tilde{s})$.

**Algorithm 1** Conservative Heuristic Computation

```
1: $c_{min} = \min_{\forall(\tilde{s},\tilde{s}')\in\tilde{E}} c(\tilde{s}, \tilde{s}')$
2: $n = 0$
3: for every $(\tilde{s}, \tilde{s}')$ do                         ▷ Compute $|\tilde{E}_{co}|$
4:     if is_conservative($(\tilde{s}, \tilde{s}')$) == True then
5:         $n = n + 1$
6: $|\tilde{E}_{co}| = n$
7: $g(\tilde{s}_g) = 0$
8: OPEN = $\tilde{s}_g$, CLOSED = $\emptyset$
9: while at least one $\tilde{s} \in \tilde{S}$ hasn't been expanded and OPEN $\neq \emptyset$
   do
10:     remove $\tilde{s}$ from OPEN with minimum $g(\tilde{s})$
11:     insert $\tilde{s}$ into CLOSED
12:     for every predecessor $\tilde{s}'$ of $\tilde{s}$ s.t $\tilde{s}$ not in CLOSED do
13:         if is_conservative($(\tilde{s}, \tilde{s}')$) == True then
14:             $c_m(\tilde{s}, \tilde{s}') = c_{min}/|\tilde{E}_{co}|$
15:         else
16:             $c_m(\tilde{s}, \tilde{s}') = \alpha c_{min}$
17:         if $g(\tilde{s}) > g(s) + c_m(\tilde{s}, \tilde{s}')$ then
18:             $g(\tilde{s}) = g(s) + c_m(\tilde{s}, \tilde{s}')$
19:             Insert $\tilde{s}$ into OPEN with $g(\tilde{s})$ as key
```

We prove the following (full proofs can be found in (Chatterjee et al. 2019)):

- $h_c$ is $\alpha$ consistent.
- wA* using $h_c$ is complete, with cost of the returned solution being no more than $w.\alpha$ times optimal solution cost in G (Thm 1,3).
- The chosen cost-scheme guarantees that the longest purely conservative path has lower cost than any path with a non-conservative edge in $\tilde{G}_m$ (Thm 4).
- If a purely conservative path exists in $\tilde{G}_m$ from $\tilde{s}_{st}$ to $\tilde{s}_g$, then number of expansions made by wA* with sufficiently large w equals the length of the shortest purely conservative path from $\tilde{s}_{st}$ to $\tilde{s}_g$ (Thm 5). If not, $h_c$ will still guide the search towards the use of edges in $G$ that have conservative mappings in $\tilde{G}_m$. In doing so, wA* returns a solution with cost within the stated sub-optimality bound. However, no guarantees on the number of expansions can be provided.

It is to be noted that finding conservative edges (function `is_conservative`($(\tilde{s}, \tilde{s}')$) in Lines 3 and 13) is domain-dependent. However, in navigation-planning domains, domain-knowledge helps in computing conservative edges efficiently. In the aformentioned example of planning in $(x, y, \textit{orientation})$, given the map of the environment and the circumradius $r_c$ of the robot, an edge can be deemed conservative by checking if both the vertices comprising the edge are at a distance greater than $r_c$ from the nearest obstacle [1]. This check is an $O(1)$ operation and can be performed during the implicit construction of $\tilde{G}_m$. The next section shows how conservative edges can be computed efficiently for two navigation domains: (1) path-planning for a UAV, and (2) humanoid foot-step planning.

## Implementation and Experimental Analysis

### Path Planning for a UAV in (X,Y,Z)

We first evaluated the effectiveness of our heuristic computation for path planning in $S = (X, Y, Z)$ for a simulated omnidirectional UAV. We discretized the environment (Fig 3 bottom left), terrain (obstacles) in maroon) into $600 \times 600 \times 400$ cells. $G$ is a 3D 26-connected grid with transition costs proportional to euclidean distance between states.

**Heuristic space:** $\tilde{G}$ is defined by dropping the $z$, or, $\lambda([x, y, z]) = [x, y]$. Thus, $\tilde{G}$ is a 2D 8-connected grid. We assume a maximum flying range in z for the UAV given by $z_{max}$. We consider $\tilde{s} = [x, y] \in \tilde{S}$ as 'free' if there exists a $z < z_{max}$ for which $s = [x, y, z]$ is obstacle-free, thus ensuring Eq. (2) holds.

**Conservative Edge Computation:** For every 2D $[x, y]$ state, obstacles in $z$ are represented as the terrain elevation-map (Fig 3, example elevation values $z_e$ shown in yellow). For a 2D edge $([x, y], [x', y'])$, let $z_e$ and $z'_e$ be the respective elevations for $[x, y]$ and $[x', y']$. We compute $z_d = |z_e - z'_e|$, the absolute difference in elevation of two adjacent 2D states forming the edge. Since the 3D grid is 26-connected, a $z_d$ value of 0 or 1 implies that every $s = [x, y, z]$ projecting to

[1]Distance-Transforms are typically used to compute and store the distance of each state in a 2D or 3D grid to its nearest obstacle.

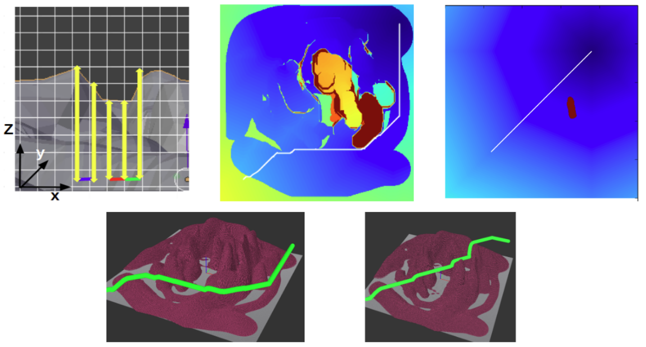

Figure 3: (top left) Cons.(red and blue) and a non-Cons. edge(green) for UAV domain, terrain elevation map (grey: terrain, yellow: elev values($z_e$)). (top center, right) visualized h-values and optimal 2D paths by $h_c$ and $h_b$ respectively. (bottom left, right) 3D paths using $h_c$, $h_b$.

$[x, y]$ has at least one collision-free edge to a $s' = [x', y', z']$ projecting to $[x', y']$, making $([x, y], [x', y'])$ conservative (Fig 3 (top left), red and blue). However, $z_d > 1$ indicates that there is one state $s = [x, y, z]$ projecting to $[x, y]$, which has no collision-free edge to any $s' = [x', y', z']$ projecting to $[x', y']$, making $([x, y], [x', y'])$ non-conservative (Fig 3 (top left), green. $z_d$ = 2).

**Results:** We compare $h_c$ with a base-line heuristic $h_b$, which is cost of the optimal path in (x,y) space but with the regular euclidean 2D edge-costs. $h_c$ is higher (yellow, red regions in Fig 3 (top center)) in the central region where elevation differences are steeper causing many infeasible 3D edges and more non-conservative edges. As a result, $h_c$ guides the search away from this region and on purely conservative paths (Fig 3(top center) white-line), unlike $h_b$ (Fig 3 (top right)). We evaluate in 20 'Easy'(gradual terrain slopes, lesser depression regions) and 20 'Difficult' (steeper slopes). Table 1 shows results. Success rate indicates the number of instances when the planner finds a solution. Heuristic computation time includes time taken to identify conservative edges and run the Backward Dijkstra's search in $\tilde{G}_m$, Planning time indicates time taken by wA* to compute a solution, total time being the sum of both. Statistics are computed for cases in which both planners were able to find a solution. $h_c$ has significantly less expansions, while producing similar solution costs and sizes. For the example in Fig 3, $h_c$ (Fig 3(bottom left)) generates a path around the steep regions, whereas $h_b$ (Fig 3(bottom right)) computes a path through the steep-sloped regions. The number of expansions using $h_c$ is 829, which is exactly equal to the solution size (829), and significantly less than the number of expansions using $h_b$ (20940). However, solution cost using $h_c$ is 8280, which is slightly greater than that using $h_b$ (6270). For the 'easy' scenarios where chances of encountering local minima are low, search using $h_c$ and $h_b$ performs similarly.

## Humanoid Footstep Planning for Bipedal Walk

For this domain, state $s = [x_l, y_l, \theta_l, x_r, y_r, \theta_r, ID] \in S$ consists of global position and orientation of the two feet

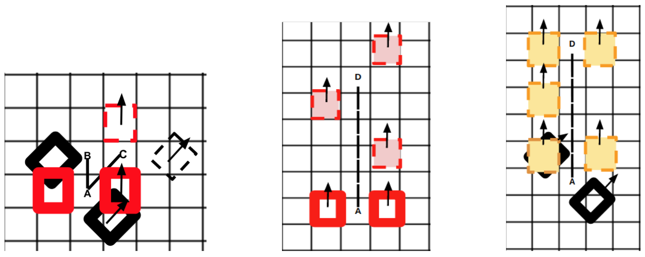

Figure 4: (left) Biped states $s_1$, $s_2$ (red, black) such that $\lambda(s_1) = \lambda(s_2) = A \in \tilde{S}$. States(pink) forming a "North" macro-move in $E$ for $s_1$. (right) States(yellow) forming a "North" macro-move in $E$ for $s_2$. $(A, D)$ is the corresponding macro-move in $\tilde{E}$.

and ID of the foot being moved next (active foot). The environment was divided into 800x800 cells. $\theta$ has a resolution of 45°. The active foot moves relative to the pivot foot. For each foot as pivot, there are 15 feasible motions of the active foot in the form of $a = [\delta x, \delta y, \delta\theta]$ relative to pivot foot. Transition-costs are proportional to euclidean distance between the feet centers.

**Heuristic Space:** $\tilde{G}$ is a 2D 8-connected grid. $\lambda$ projects the 2 feet-positions in $s$ to the 2D grid by computing their mean. Cost of an edge is proportional to the euclidean distance between 2D cell-centres.

**Conservative Edge Computation:** For an $(\tilde{s}, \tilde{s}')$ to be conservative, *every* $s \in \lambda^{-1}(\tilde{s})$ should have *at least one* valid pose leading to a state in $\lambda^{-1}(\tilde{s}')$, such that the mean moves along $(\tilde{s}, \tilde{s}')$. Owing to the kinematic constraints of this humanoid, most edges in set $\tilde{E}$ are non-conservative. Consider the example in Fig 4, where the active foot (right) can have 2 feasible motions: $a_1 = [2, 0, 0]$ or $a_2 = [0, 0, 45°]$. Let $s_1, s_2$ be states in $\lambda^{-1}(A)$, shown in Fig 4 (left) in red and black respectively. None of the actions $a_1$ or $a_2$ applied in $s_2$ result in a successor that projects to $B$. Similarly, no action applied in $s_1$ results in a successor that projects to $C$. Thus, $(A, B)$ and $(A, C)$ are both non-conservative edges.

However, consider the state $D$, 5 edges North of $A$ in the heuristic-space (Fig 4 (center)). Applying $a_1$ thrice from $s_1$ (Fig 4 (center)) moves the mean to $D$. Applying $a_2$ and then $a_1$ four times from $s_2$ (Fig 4 (right)) also moves the mean to $D$. Thus, we can choose a state $\tilde{s}'$, $k$ edges away from $A$ in heuristic space, with $k$ being sufficiently large such that *every* $s \in \lambda^{-1}(A)$ has a sequence of valid steps in $G$ to move the mean to $\tilde{s}'$. We call this sequence of moves as a 'macro-move'. Macro-moves exist in $G$ as well as $\tilde{G}$. Fig 4 (center) and Fig 4 (right) depict macro-moves in $G$ from $s_1$ and $s_2$ respectively, corresponding to the macro-move $(A, D)$ in $\tilde{G}$. If we can find macro-moves from *every* $s \in \lambda^{-1}(A)$ to some $s' \in \lambda^{-1}(D)$, then $(A, D)$ satisfies the conservative property and thus, becomes a conservative macro-move.

We can generate such macro-moves from $A$ in other directions as well. Here, we select $k = 20$ and add macro-moves in 4 directions (North, East, West, South) in $\tilde{G}$. These macro-

| type of environment | heuristic | w in wA* | succ rate | # expansions | sol size | sol cost | planning time(s) | heur comp time(s) |
|---|---|---|---|---|---|---|---|---|
| difficult | $h_c$ | 100 | 100% | 534±99 | 534±99 | 5333±994 | 0.002±0.001 | 0.21±0.003 |
| difficult | $h_b$ | 100 | 80% | 56687±30740 | 537±94 | 5357±943 | 0.028±0.03 | 0.21±0.01 |
| easy | $h_c$ | 100 | 100% | 894±1263 | 381±96 | 3798±969 | 0.001±0.001 | 0.21±0.009 |
| easy | $h_b$ | 100 | 100% | 1921±4300 | 346±112 | 3452±1160 | 0.002±0.002 | 0.20±0.05 |

Table 1: Comparison of $h_c$ with baseline $h_b$ for UAV domain in 'difficult' (more local minima) and 'easy' scenarios

| heuristic | w in wA* | succ. rate | # expansions | sol size | sol cost | planning time(s) | heur comp time(s) |
|---|---|---|---|---|---|---|---|
| $h_c$ | 100 | 37/39 | 1720±5850 | 50±35 | 156250±110820 | 6±19.8 | 0.07 ±1.39 |
| $h_b$ | 100 | 24/39 | 16000±19000 | 61.1±62.8 | 51630±52396 | 36±43 | 0.036±0.006 |

Table 2: Comparison of $h_c$ with baseline $h_b$ in Footstep Planning.

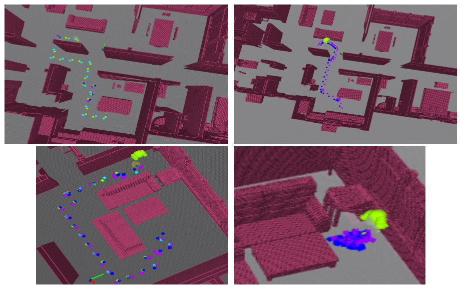
Figure 5: Results with $h_c$ (left) and $h_b$ (right) for footstep planning.

moves represent sequence of actions and are analogous to motion primitives and can be computed offline. Once computed for all $s \in \lambda^{-1}(A)$ for any $A \in \tilde{S}$, these can be applied to any state $s \in S$ to compute its relevant successors.

We compute conservative heuristic $h_c(s)$ of state $s$, by performing a Dijstra's search in $\tilde{G}$ with the macro-moves included. Note that, macro-moves are like any other edge and no edges have been removed from any of the graphs, hence all the guarantees described before still hold.

**Results (Simulation):** We used the model of a full-size humanoid for our experiments and allowed a maximum planning time of 90 seconds. We chose a typical indoor environment with narrow passages, corridors and varying doorway sizes. Also, the environment has multiple pathways for traversing between different locations, therefore requiring the search to explore these options. Table 2 compares the performance of conservative heuristic ($h_c$) with the baseline 2D Dijkstra's shortest path heuristic ($h_b$). Results are averaged over 39 random starts and goals. $h_c$ has a remarkably higher success rate (94.8%) compared to $h_b$ (61.5%). Moreover, using $h_c$ reduces planning times by an order of $\approx$ 6 and expansions in the final search by a factor of 10. The heuristic computation time is slightly higher for $h_c$ (0.07s) than $h_b$ (0.036s) owing to the addition of macro-moves in $\tilde{E}$. Two examples of the search using $h_c$ and $h_b$ shown in Fig 5 (left) and (right) respectively. In the first example (top), wA* with $h_b$ takes a long time but finds a path through the narrow passage. In the second example (bottom), wA* with $h_b$ isn't even able to find the solution in 90s.

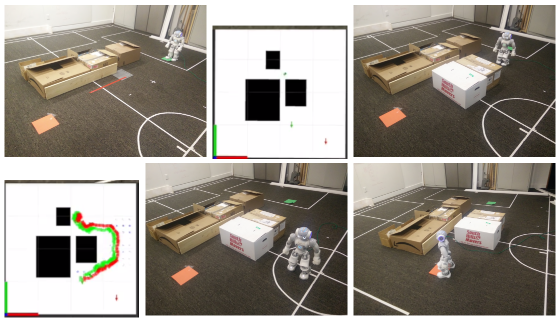
Figure 6: Initial environment (top left). Planner stuck in local minima (top center) created by the narrow passage between white and brown box using $h_b$, NAO waiting for plan (top right). NAO executing plan computed using $h_c$ (bottom).

**Results (Robot experiment):** Fast re-planning is useful when the environment is changing. We implemented a footstep planner for the NAO robot based on (Hornung, Maier, and Bennewitz 2013) and show advantages of using $h_c$ in quickly computing footstep plans in the following scenario:

- We placed a set of static obstacles in the environment and computed initial footstep plans using $h_c$ and $h_b$.
- During execution, we added an obstacle in the environment creating a narrow passage on the robot's path, triggering re-planning. wA*(w=8) using $h_b$ keeps expanding states in the local minima created by the passage and fails to find a path in the given time. wA*(w=8) using $h_c$ re-plans in 978 ms (heuristic computation time: 187 ms, planning time: 791 ms). A video demonstration of this experiment can be found here: https://youtu.be/zs4HX84jE1w

## Conclusion

We proposed a heuristic computation algorithm using conservative edges in the heuristic-space for reducing state expansions by wA*, proved theoretical properties and applied our approach in several motion planning domains, observing significant reductions in expansions and planning times. Future work is to have a probabilistic definition of the conservative property.

## References

Bacchus, F., and Yang, Q. 1991. The downward refinement property. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'91, 286–292. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Bäckström, C., and Jonsson, P. 2013. Bridging the gap between refinement and heuristics in abstraction. In *IJCAI*, 2261–2267.

Bulitko, V.; Sturtevant, N. R.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *J. Artif. Intell. Res.(JAIR)* 30:51–100.

Chatterjee, I.; Likhachev, M.; Khadke, A.; and Veloso, M. 2019. Speeding up search-based motion planning via conservative heuristics. Tech. Report, The Robotics Institute, Carnegie Mellon University.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Computational Intelligence* 14(3):318–334.

Ferguson, D.; Likhachev, M.; and Stentz, A. 2005. A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, 9–18.

Haslum, P., et al. 2007. Reducing accidental complexity in planning problems. In *IJCAI*, 1898–1903.

Helmert, M.; Haslum, P.; Hoffmann, J.; et al. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996. Speeding up problem solving by abstraction: A graph oriented approach. *Artificial Intelligence* 85(1-2):321–361.

Hornung, A.; Maier, D.; and Bennewitz, M. 2013. M.: Search-based footstep planning. In *In: ICRA Workshop Progress and Open Problems in Motion Planning and Navigation for Humanoids*.

Hwangbo, M.; Kuffner, J. J.; and Kanade, T. 2007. Efficient two-phase 3d motion planning for small fixed-wing uavs. *Proceedings 2007 IEEE International Conference on Robotics and Automation* 1035–1041.

Liu, S.; Mohta, K.; Atanasov, N.; and Kumar, V. 2018. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters* 3(3):2439–2446.

Pang, B., and Holte, R. C. 2011. State-set search. In *Fourth Annual Symposium on Combinatorial Search*.

Pearl, J. 1984. Heuristics: intelligent search strategies for computer problem solving.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial intelligence* 1(3-4):193–204.

Vega-Brown, W., and Roy, N. 2018. Admissible abstractions for near-optimal task and motion planning. *arXiv preprint arXiv:1806.00805*.

Wilt, C. M., and Ruml, W. 2012. When does weighted a* fail? In *SOCS*, 137–144.