Search Reduction Through Conservative Abstract-Space Based Heuristic

Ishani Chatterjee, Maxim Likhachev, Manuela Veloso
Carnegie Mellon University

Abstract

The efficiency of heuristic search depends dramatically on the quality of the heuristic function used. For an optimal heuristic search, heuristics that estimate cost-to-goal better typically lead to faster search times. For a sub-optimal heuristic search such as weighted A* on the other hand, the search speed depends more on the correlation between the heuristic function and the true cost-to-goal function. In this extended abstract, we discuss our preliminary work on computing heuristic functions that exploit this fact and aim to reduce the number of states expanded by weighted A* search before it finds a path to the goal.

Introduction and Problem Description

It is desired to have a heuristic function that leads the search to the goal with minimum number of expansions possible. To achieve this, it is common for a heuristic function to be computed on a graph derived by abstracting the original graph. Cases have been shown (Wilt and Ruml 2015) where heuristics designed for optimal searches may not be suited for sub-optimal searches and a strong correlation of the heuristic with the true cost-to-goal or the node-distance-to-goal (Wilt and Ruml 2012) can result in a quicker search by reducing inefficient expansions, defined as the expansion of nodes that do not appear in the final path given by the search. A weak correlation may lead to a local minima and a lot of inefficient expansions. This motivates us to come up with a heuristic function that serves the aforementioned purpose. Our contribution is in defining the notion of $conservative$ and $non-conservative$ edges. Two abstract states are connected by a $conservative$ edge in the abstract space, only if all corresponding pairs of states in the original state space are also connected via a direct edge. In the context of weighted A* with a sufficiently large inflation of heuristic, a heuristic function that leads the search along the paths in the abstract space that maximize the usage of the conservative edges results in the search that is more likely to find a path to the goal without getting stuck in local minima.

The planning problem is defined as searching a graph G = (S,E) where S is the set of states, and E is the set of edges in the graph. (s, s') denotes an edge in E connecting states s, $s' \in S$. We assume all edge costs are finite. The objective of the search is to find a path from state s_{start} to the goal state s_g in G. To illustrate some concepts, we will use the 3D navigation planning domain with the robot having three degrees of freedom,i.e, $S = \{(x, y, \theta(heading)) \mid x \in \{1, ..., n\}, y \in \{1, ..., m\}, \theta \in \{1, ..., 8\}\}$, where n = width and m = height of the environment represented as an 8-connected grid. We assume that the robot can turn in place and can move in any direction on the 8-connected grid. Let $\lambda : S \rightarrow \tilde{S}$ be the many-to-one mapping representing the abstraction of each state in S to the abstract space $\tilde{S}$, defined by:$\lambda(s) = \tilde{s}$, where $s \in S$, $\tilde{s} \in \tilde{S}$ s.t $|\tilde{S}| \leq |S|$. λ^{-1} is the inverse one-to-many mapping from $\tilde{S}$ to S defined by: $\lambda^{-1}(\tilde{s}) = \{s \in S \mid \lambda(s) = \tilde{s}\}$ For example, in the 3D navigation domain we use the mapping $\lambda((x, y, \theta)) = (x, y)$, by dropping the third degree of freedom θ.

The abstract space is also a graph $\tilde{G} = \{\tilde{S}, \tilde{E}\}$. Each edge has a flag indicating if it is $conservative$ or not. An edge $(\tilde{s}, \tilde{s}')$ for $\tilde{s}, \tilde{s}' \in \tilde{S}$ is $conservative$ if the following can be guaranteed: $\forall s \in \lambda^{-1}(\tilde{s})$, $\exists$ at least one $(s, s') \in E$ s.t $s' \in \lambda^{-1}(\tilde{s}')$. For example, in 3D navigation domain, $\tilde{G}$ is an 8-connected grid in which the obstacles in the map are inflated by the inscribed radius of the robot footprint to exclude those states where the robot center cannot physically lie. The edges that are labeled as conservative on the other hand are those that connect cells that remain to be valid for all possible orientations of the robot. To determine which edges are conservative, we construct a *conservative* representation of the space, also a graph, as an 8-connected grid in which we inflate the obstacles by the radius of the circle that circumscribes the footprint. More generally, a conservative graph is a subset of $\tilde{G}$ that consists purely of conservative edges. Given G and $\tilde{G}$ with edges flagged as $conservative$, we aim to compute a heuristic function h: $S \rightarrow \mathbb{N}$ that reduces the efforts weighted A* performs to find a path to the goal.

Heuristic computation

Ideally, for a sufficiently large weight of the heuristic in weighted A* search, if $h(s)$ can make the weighted A* search expand only those successors $s \in S$ of state $s' \in S$ for which it holds that $(\lambda(s), \lambda(s'))$ is a conservative edge in the abstract space then the search would always be pro-

ceeding towards the goal without getting stuck in local minima. However, sometimes such paths to the goal do not exist. We therefore want to compute $h(s)$ such that it guides the search in the original space along the paths that minimize the number of non-conservative edges in the abstract space. We first compute a heuristic cost-to-goal estimate $\tilde{h}(\tilde{s}) \forall \tilde{s} \in \tilde{S}$ in the abstract space, and use $h(s) = \tilde{h}(\lambda(s))$ for all s in S. To compute $\tilde{h}(\tilde{s})$, we re-create the cost of an edge between $\tilde{s}$ and $\tilde{s}'$, denoted by $C(\tilde{s}, \tilde{s}')$, such that $non-conservative$ edges are more expensive than $conservative$ ones. $C(\tilde{s}, \tilde{s}')$ is defined as: $C(\tilde{s}, \tilde{s}') = \gamma$ if $(\tilde{s}, \tilde{s}')$ is $conservative$, $m\gamma$ otherwise, where $\infty > \gamma > 0$ and $\infty > m > 1$.The heuristic function is then given by: $\forall \tilde{s} \in \tilde{S}$, $\tilde{h}(\tilde{s}) = min_{\tilde{\pi}(\tilde{s}, \lambda(s_g))} C(\tilde{\pi}(\tilde{s}, \lambda(s_g)))$, where $\tilde{\pi}(\tilde{s}, \lambda(s_g))$ is a path from $\tilde{s}$ to a goal state in the abstract space and $C(\tilde{\pi}(\tilde{s}, \lambda(s_g)))$ is the cost of this path using the cost function described above. The heuristic function can be computed using a single backward Djisktra's search in the abstract space starting at $\lambda(s_g)$, with edge costs set as γ for $conservative$ and $m\gamma$ for $non-conservative$ edges. Thus, in our 3D navigation example, the heuristic is computed by running a backward Dijktra's search on 2D grid with the cost of transitions set to γ whenever the edge connects two cells that are both in the conservative representation (that is, valid in the map after the obstacles are inflated by the radius of the circle circumscribing the footprint of the robot) and $m\gamma$ otherwise. Assuming the abstract graph $\tilde{G}$ was constructed by relaxing the original problem, it satisfies the property that if state $s \in G$ has a finite length path to the goal, then $\tilde{s}$ also has a finite length path to $\tilde{s_g}$. Given this, it is obvious that weighted A* search using our heuristic function is complete. It is so, because every state $s \in G$ with a finite cost path to the goal will have a finite $h(s)$ as long as γ and m are finite. Unfortunately though, $h(s)$ can be inadmissible, and it is future work to analyze the degree to which it can be inadmissible and how this inadmissibility can be controlled.

Experimental Setup, Results and Discussion

We used the 3D navigation domain for evaluations on 4 environments representing turns in corridors, wide and narrow passages and doorways: Env1,Env2,Env3 had $301 \times 299 \times 8(x \times y \times \theta)$ states. Env4 had $2211 \times 1947 \times 8$ states. A rectangular robot footprint having 0.2m and 0.3m as inscribed and circumscribed radii was used in Env4. For others 0.3m and 0.4m was used. The abstract state-space, conservative representation and visualized heuristics are shown in Fig 1. We set $\gamma = 1$ and $m = 10$. We used weighted A* in the original space, with a weight of 10000. We compared the performance of $h(s)$ with another heuristic, $h'(s)$, which computes the optimal cost-to-goal in the abstract space without the notion of $conservative$, using the usual transition costs in the 8-connected 2D Grid. Results of the comparison are shown in Figure1 and Table 1. The solution size and quality is slightly sub-optimal, but there is a large reduction in expansions. The number of expansions using h(s) is linear in solution size for Env1,2 and 3, because the search expands solely through the conservative region. $h'(s)$ enters the local minima and therefore has more expansions. For Env4,

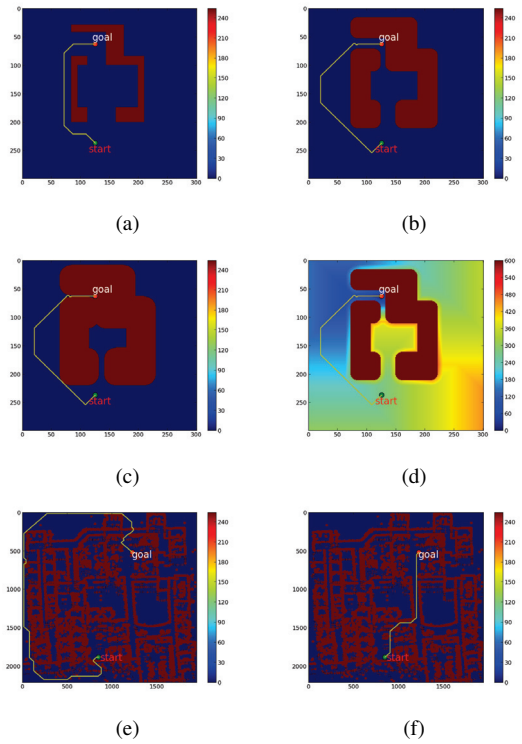

Figure 1: Env1:(a) Final path using $h'(s)$, (b) abstract space, final path using h(s), (c) conservative rep. (d) visualized heuristics. Env4: (e) path using $h(s)$, (h) using $h'(s)$.

environment	heuristic	no. of expansions	solution size	solution cost
env1	$h(s)$	392	274	23747
	$h'(s)$	59533	250	22509
env2	$h(s)$	543	281	32922
	$h'(s)$	179435	211	13150
env3	$h(s)$	489	270	32240
	$h'(s)$	320613	244	22392
env4	$h(s)$	2990146	4436	206101
	$h'(s)$	11569828	1491	55899

Table 1: Comparison between conservative and non-conservative heuristics

that does not have a large number of conservative edges, the number of expansions for h(s) is still reduced compared to $h'(s)$.

References

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

Wilt, C. M., and Ruml, W. 2015. Building a heuristic for greedy search. In *Eighth Annual Symposium on Combinatorial Search*.