Bootstrapping Abstract Planning with Experience Graphs

Aram Ebtekar, Maxim Likhachev, Mike Phillips, Sven Koenig
Carnegie Mellon University
5000 Forbes Ave,
Pittsburgh, PA 15213

Abstract

A* and variants are regularly applied to plan paths in a graph. Rather than planning each query from scratch, we follow the recent trend across AI and robotics of collecting and reusing information, i.e. learning. Experience-graphs (E-graphs) store past plans in order to speed up related planning episodes in the future. New challenges arise when we try to apply E-graphs in high-level planning, where the state graph is implicitly exponentially sized in its description length. We describe an algorithm that combines E-graphs with a standard STRIPS relaxation to form ϵ-admissible heuristics in a domain-independent manner.

Introduction

Planning entails finding action sequences that achieve a goal state. In order to apply the language of graph theory, states are often thought of as nodes connected by actions. A *plan* is a path from start to goal. Despite the existence of classic polynomial-time search techniques, such as Dijkstra's algorithm, planning remains the crux of many difficult problems in AI. The reason usually is that the graphs involved are very large, in fact so large that they cannot be stored explicitly. By either observing new states or using a compressed representation of the graph's structure, states may be generated dynamically as the search progresses. In motion planning, the space may be a continuous geometric set which is dynamically discretized.

In high-level logical planning, a domain language such as STRIPS, SAS+ or PDDL encode a graph whose size is exponential in its description length. These short descriptions often create structure that can be exploited. In order to create a domain-independent solver for a language such as STRIPS, we need to automatically extract and interpret information about the problem.

We may view planning as a process that generates distance-to-goal labels for each node of a graph. Given these distance labels, it's a simple matter to generate a plan by tracing steps from the goal, successively taking the optimal predecessor. Since there is no worst-case polynomial-time algorithm that can solve the shortest-paths problem in STRIPS (recall the graph is exponential-sized), we may try to roughly approximate the distance labels. The approximate labels we call **heuristics**, and we use them in hopes of finding plans more efficiently. If the heuristic precisely reflects the true distances, we saw how to immediately find a path. Algorithms such as A* are designed to find solutions quickly whenever the heuristic is of reasonable quality. Note that in practice, any subexponential algorithm can only afford to look at a subset of the nodes, so it is only for these that we compute the heuristic. If the heuristic is good, we hope not to have to look at (i.e. generate) too many nodes. Hence, there is a tradeoff between spending more time to get a tighter heuristic, vs evaluating a weaker heuristic more quickly but at many more nodes.

In practice, even with a domain-independent means of generating heuristics, many STRIPS problems can be hard. Even humans have difficulty when faced with an unfamiliar kind of puzzle, though we get better with experience. Thus, one would hope that a planning agent would similarly learn to generalize solutions from past planning instances to related new instances. A recent approach builds *Experience graphs* to represent the high-level connectivity of the free space in motion planning tasks. The intuition is to remember previously generated paths so that, when a new start and goal is queried, a new path can be more quickly generated by reusing subpaths from the E-graph. However, E-graphs have never before been applied to high-level representations such as STRIPS. This is our present contribution.

In section [???], we present the HSP heuristic for STRIPS. Then, we discuss E-graphs in more technical detail. This paper's novel contributions follow thereafter, where we extend E-graphs to STRIPS planning by fitting it on top of the HSP heuristic. We prove a few theoretical properties of this algorithm, most notably its ϵ-optimality guarantee, and then present experiments showing domains in which this method achieves significant speedups over plain HSP.

Related Work

HSP

A STRIPS problem is a tuple $P = \langle A, O, I, G \rangle$ consisting of an atom set A, operator set O, initial state $I \subset A$ and goal condition $G \subset A$. Each operator $op \in O$ is defined by its cost and its precondition, add and delete lists: $Cost(op) \in$

$\mathbb{R}^+$ and $Prec(op), Add(op), Del(op) \subset A$.

The STRIPS problem P defines a directed state graph (V, c) where V is the power set of A (i.e. states $v \in V$ correspond to collections of atoms), and

$$c(u,v) = \inf\{Cost(op) \mid u \in Prec(op), \\ v = u - Del(op) \cup Add(op)\}$$

Naturally, $c(u,v) = \infty$ when there is no operator directly linking u to v. Given a STRIPS problem P, we seek a low-cost path from the initial state $I \in V$ to any of the goal states $g \in V$ such that $g \supset G$.

Many of today's state-of-the-art domain-independent STRIPS planners are based on the HSP planner, which we now describe. It's common to compute heuristics by solving an easier, relaxed version of the original problem. In STRIPS, one might ignore the delete lists of operations. Since having more atoms makes preconditions and the goal condition more likely to hold, this can only make the problem easier, so the resulting heuristic is admissible.

However, even the relaxed STRIPS planning problem includes subset-sum as a special case, making it NP-hard. Intuitively, we see that the search space is still exponential. To simplify things, we decouple the atoms, instead estimating the cost to achieving each individual atom.

Let's say we want to compute a distance-to-G heuristic for a state S. Estimate the cost to achieve an atom $a \in A$ from S by $g_S(a) =$

$$\begin{cases} 0 & \text{if } a \in S \\ \min_{op, a \in Add(op)} \left(g_S(Prec(op) + Cost(op))\right) & \text{if } a \notin S \end{cases}$$

Define the HSP-max heuristic by $h^{HSP}(S) = g_S(G)$, the estimated cost of achieving all atoms in G. In order to cheaply compute $g_S(G)$ and $g_S(Prec(op))$, we define $g_S(P) = \max_{p \in P} g_S(p)$ for sets of atoms P. This admissible and consistent. If the atoms were truly independent, a much tighter estimate would be $g_S(P) = \sum_{p \in P} g_S(p)$. Despite its inadmissibility in general, the latter HSP-plus heuristic is often useful in practice, as it uses more information and biases more greedily toward the goal.

From a computational perspective, the HSP heuristic must compute $g_S(a)$ for all $a \in A$ whenever a new state S is generated. This can be done by dynamic programming, essentially searching forward from the atoms in S. In a sense, it seems we are wasting a lot of work by estimating the cost to reach every atom when we only require $g_S(G)$. Later, we'll see how to make fuller use of this computation.

Experience Graphs

E-graphs are used to optimize search time over multiple instances on the same graph. Between instances, we update the set of edges which constitute the E-graph, typically by adding the previous search's resulting path. If the heuristic search is done on a reduced state space by projection, the E-graph is considered as a subgraph of the latter.

Suppose we are given a consistent heuristic $h(u,v)$ for the distance between arbitrary pairs of states. That is, h obeys the triangle inequality with respect to edge costs, and $h(g) = 0$ for every goal state g. In order to derive a related heuristic which biases in favor of using the E-Graph, we inflate the heuristic by a factor $\epsilon^E > 1$, but allow it to use uninflated "shortcuts" on the E-graph. To be precise, the E-graph heuristic is defined by

$$h^E(S) = \min_{\pi} \sum_i \min\{\epsilon^E h(s_i, s_{i+1}), c^E(s_i, s_{i+1})\}$$

over all paths $\pi = \langle s_0, s_1, ..., s_N \rangle$ with $s_0 = S$ and $s_N = G$.

In the limit as $\epsilon^E \to 1$, the "shortcuts" become pointless, so

$$h^E(S) = \min_{\pi} \sum_i h(s_i, s_{i+1}) = h(s_0, s_N) = h(S, G)$$

by the triangle inequality. Conversely as $\epsilon^E \to \infty$, the minimizing path becomes the one which uses the fewest edges outside the E-graph, and the cost remains finite iff S and G are in the same E-graph component.

We cannot afford to compute h^E according to its literal definition, as there are too many paths to consider. Fortunately, there is a practical means of computing it. By the triangle inequality, any pair of consecutive $h(s_i, s_{i+1})$ terms can be combined into one. Thus, we only need be concerned with states s_i lying on the E-graph for $0 < i < N$.

Let V^E consist of the goal state G plus all endpoints of the E-graph's edges. In a preprocessing stage before a search, we can apply Dijkstra's algorithm once in reverse to compute the estimated distance $h^E(S)$ to G from every point $S \in V^E$, using any combination of E-graph edges and "jumps" with cost $\epsilon^E h(s_i, s_{i+1})$. This runs in $O(|V^E|^2)$ time, which is negligible when the E-graph is small.

Later when expanding $S \notin V^E$, we see that we've already precomputed all but the first leg of the journey in the definition of $h^E(S)$. Thus, we derive the simpler computation

$$h^E(S) = \min_{S' \in V^E} \left(\epsilon^E h(S, S') + h^E(S')\right)$$

h^E is ϵ^E-consistent [cite].

Algorithm Discussion and Analysis

Now we combine the ideas from HSP and E-graphs to construct a domain-independent STRIPS planner which learns from experience. In STRIPS, the consistent heuristic $h(u,v)$ is given by the HSP-max estimate $h_u^{HSP}(v)$. Thus, we can compute $h_u^{HSP}(v)$ for all $u, v \in V^E$ in precisely $|V^E|$ runs of the dynamic programming we saw before. Thus, our preprocessing work is roughly equivalent to that of generating $|V^E|$ states, which will be small. As before, Dijkstra's algorithm combines an inflation of these estimates with the E-graph edges themselves to derive $h^E(S)$ for all $S \in V^E$.

Upon encountering a new state $S \notin V^E$, we use dynamic programming as before to compute $g_S(a)$ for all atoms $a \in A$. Then it's a simple matter to compute the E-graph heuristic as

$$h^E(S) = \min_{S' \in V^E} \left(\epsilon^E g_S(S') + h^E(S')\right)$$

Here we see that, when E-graphs are involved, $g_S(S')$ is computed for many states S', not just the goal. Thus, we consult the E-graph essentially for free since HSP would compute all the g-values anyway. The only additional work here is the minimization over $S' \in V^E$, which is negligible for small E-graphs.

We should remark that, although STRIPS problem as we defined them have multiple goal states defined by the goal condition G, the heuristic's monotonicity implies that the set of atoms precisely matching G will be the lowest-cost goal to reach in the heuristic approximation.

Algorithm 1 Search()

repeat
  for all $a \in A$ **do**
    if $a \in S$ **then**
      $g_S(a) = 0$
    else
      $g_S(a) = \min_{op, a \in Add(op)} (g_S(Prec(op) + Cost(op)))$
    end if
  end for
until nothing changed

Experiments

Conclusion and Extensions