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 are never stored explicitly. Pieces may be generated dynamically as the search progresses, using a short description of the graph's structure. In motion planning, the space is often a continuous geometric set which is discretized during the search. In high-level logical planning, a domain language such as STRIPS or SAS+, or the more elaborate PDDL, is used. These short descriptions put structural constraints on the graph, which may be used to aid the search.

In order to create a domain-independent solver for a language such as STRIPS, we need to extract information about the problem automatically.

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 successively moving to 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 of course find a path immediately. 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. Thus, 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.

However, even with a domain-independent heuristic, many STRIPS problems can be hard. Even humans have difficulty when faced with an unfamiliar puzzle. However, one would hope that a planning agent would learn to generalize solutions from past planning instances to similar new instances. A recent approach builds *Experience graphs* to represent the high-level connectivity of the free space in motion planning tasks. The high-level idea is to remember previously generated paths so that, when a new start and goal is queries, a new path can be more quickly generated by reusing pieces of the E-graph.

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

Related Work

HSP

A STRIPS problem is a tuple $P = \langle A, O, I, G\rangle$ of atoms A, operators O, an initial state $I \subset A$ and goal state $G \subset A$. Each operator $op \in O$ has precondition, add and delete lists: $Prec(op), Add(op), Del(op) \subset A$. The STRIPS problem P defines a state space $\mathcal{S} = \langle S, s_0, S_G, \rangle$.

To find optimal paths, we need a heuristic which underestimates true distances. A simple way to do this is to solve a relaxed version of the original problem, e.g. by ignoring certain constraints. The STRIPS representation is monotone, so one possible relaxation is to simply ignore the delete lists of operators. However, this relaxation is still hard to solve, as it includes subset-sum as a special case. One way to ensure a fast calculation is to cast aside the exponential state space by estimating the costs for achieving individual atoms. These costs can then be combined by taking their sum if the atoms are independent or, to ensure consistency in the general case, we can take their maximum. Thus, $h(S) = g_S(G)$ where

$$g_S(a) = \begin{cases} 0 & \text{if } a \in S \\ 1 + \min_{op, a \in Add(op)} g(Prec(op)) & \text{otherwise} \end{cases}$$

for atoms $a \in A$, and $g_S(P) = \max_{p \in P} g_S(p)$ for sets of atoms P.

Experience Graphs

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

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

Algorithm Discussion and Analysis

For $S \in N^E$, we compute $h^E(S)$ by fusing the E-graph edges and off-E-graph estimates with Dijkstra's algorithm. The complexity of this pre-processing step is quadratic in the number of E-graph nodes.

For $S \notin N^E$,

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

Experiments

Conclusion and Extensions