Bootstrapping Abstract Planning with Experience Graphs

Aram Ebtekar, Maxim Likhachev
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

asdf

Algorithm

When planning over possible states of the world, it's typically more efficient to describe the states in terms of a set of features. The set of states is exponential in the number of features, so even a small problem description may imply a large graph. In general, planning over such a large space is computationally infeasible.

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 cheapest neighbour. Since there is no deterministic polynomial-time algorithm that can solve the shortest-paths problem in STRIPS (recall the graph is exponential-sized), we may try an approximation algorithm. The resulting labels we call **heuristics**, and we use them in hopes of achieving optimal or near-optimal plans more efficiently. If the heuristic precisely reflects the true distances, we find a path immediately. Algorithms such as A* are designed to find solutions quickly whenever the heuristic is reasonably tight and **ϵ-consistent**. 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 tight heuristic, vs evaluating a weaker heuristic more quickly but also more frequently.