Relaxed Decision Diagrams for Delete-Free Planning

Augusto B. Corrêa and **Florian Pommerening** and **Guillem Francès**
University of Basel, Switzerland
{augusto.blaascorrea, florian.pommerening, guillem.frances}@unibas.ch

Introduction

In classical planning, we want to find a sequence of actions that leads to a state satisfying a goal condition given only an initial representation of the world and a set of possible actions. One technique to solve classical planning problems is *heuristic search*. In heuristic search, a *heuristic function* h maps each state to a value estimating its distance to the closest goal state. A heuristic guides the search through the space of all possible world states, by exploring different reachable states, until it reaches a state satisficing the goal. If the quality of the estimation of h is good enough, the planner can mitigate the state explosion problem. If heuristic values are lower bounds to the true goal distance (i.e., if the heuristic is *admissible*), A* search can be used to find optimal solutions.

A common way to represent a planning problem is to use a factored representation of the world, where each state is represented by a set of *facts* and actions can *add* and *delete* facts from a state. Here we consider a fragment of classical planning called *delete-free planning* where facts that are made true remain true forever. In other words, an action can only add facts to a state, but never delete them. Finding optimal solutions to delete-free planning tasks remains NP-hard (Bylander 1994) but is interesting because many successful heuristics are based on a delete-free relaxation of a classical task (e.g., Helmert and Domshlak 2009) and some classical tasks are inherently delete-free (e.g., Gefen and Brafman 2011).

We present a way to extract lower bounds for delete-free tasks from *relaxed decision diagrams* (RDDs) (Andersen et al. 2007; Bergman et al. 2016). We show how the set of all plans can be represented by a decision diagram. The relaxed version of this decision diagram then overapproximates the set of all plans and can be used to generate lower bounds in polynomial time. The lower bounds can be used as admissible heuristic values to find optimal solutions for such tasks.

We discuss different ways to use RDDs to solve delete-free planning tasks and investigate how close the obtained lower bounds are to the optimal solution.

Background

Delete-free Planning In a *delete-free planning task*, the world is described by a set of *facts* P. States are subsets $s \subseteq P$ with the interpretation that the facts in s are true and all other facts are false. The *initial state* of the world is given as a state s_0 and the *goal* of the task is given as a set $\gamma_* \subseteq P$ of facts that have to be true in a goal state. A set of *actions* A describes how the world can be affected. Each action $a \in A$ is associated with a *cost* $c(a) \in \mathbb{N}$, a *precondition* $\text{pre}_a \subseteq P$, and an *effect* $\text{add}_a \subseteq P$. It is *applicable* in a state s if its precondition is satisfied ($\text{pre}_a \subseteq s$) and applying a in s leads to the state $s[a] = s \cup \text{add}_a$. A sequence of applicable actions $\langle a_1, a_2, \ldots, a_n \rangle$, is called a *plan* if it ends in a goal state, i.e., $s_0[a_1][\ldots][a_n] \subseteq \gamma_*$. If it has least cost among all plans, it is called an *optimal plan* and its cost is denoted as h^+.

Decision Diagrams *Decision diagrams* (DD) provide an efficient way to encode solutions of optimization problems with factored representations. A DD is represented by a directed acyclic graph G, divided in layers $L_1, \ldots, L_n$ where L_1 contains only the root node r and L_n has only the terminal node t. Each node g in layer L_i, $i < n$ is associated with a variable v of the optimization problem. There is an edge from g to a node in layer L_{i+1} for every possible value d of v labeled with $v := d$. A path in G can thus be seen as a set of assignment of variables to their values. Each node represents the set of partial assignments represented by all paths leading to it from r. The set of assigments represented by node t is called the set of *solutions* of the DD. Costs can be represented by associating a weight with each edge in the DD. The cost of a path then is the sum of costs of its edges and the cost of a node is the cost of the cheapest path leading to it. A DD for a given problem $\mathcal{P}$ is an *exact decision diagrams* (EDD) if all solutions of $\mathcal{P}$ are represented as a path from r to t with same cost, and vice versa.

For NP-hard problems, constructing an EDD is impossible in polynomial time (unless P = NP). However, it is possible to efficiently construct a *relaxed decision diagram* (RDD) that overapproximates the solutions of the EDD. To do so, we construct the RDD layer by layer, starting from the root node. During construction, we limit the number of nodes in each layer (called its *width*) to a constant ω. If a layer has more than ω nodes, we *combine* a subset of nodes in this layer into a single new node until there are only ω nodes left. If the combination satisfies the following two properties, the RDD overapproximates the EDD: (i) no solution for the original EDD can be discarded; and (ii) no solution represented by a path can have its cost overestimated.

RDDs for Delete-Free Planning

We want to construct an RDD that overapproximates the solutions of a delete-free planning task Π. We first show how an EDD for a given task can be constructed, then explain how nodes are selected and combined to construct an RDD.

In the construction, the nodes of the decision diagram roughly correspond to states of Π. Each node g is associated with two properties: the facts $F(g) \subseteq P$ that are already reached, and a set of actions $N(g) \subseteq A$ that should not be used in states reached from g. In the root node r, the facts of the initial state are reached ($F(r) = s_0$) and no action is forbidden ($N(r) = \emptyset$). In each node g the EDD then branches over an action a that can be applied in this state ($\text{pre}_a \subseteq F(g)$), adds at least one reached fact ($\text{add}_a \not\subseteq F(g)$), and is not forbidden ($a \notin N(g)$). The decision for a is whether to apply it right now, or never in any state reached from g. This choice is possible because once an action is applicable in a delete-free tasks, it remains applicable and applying it later cannot be beneficial. In the successor where a is applied, we add add_a to the reached facts; in the successor where a is not applied, we keep the same set of facts as g. In both successors, we add a to the set of forbidden actions. If a generated node g has $\gamma_* \subseteq F(g)$, we merge it into t. If no action exists that is applicable in a node g, then g is a dead end and we prune it. We repeat this procedure with the new successors until all leaf nodes are pruned or merged this into t. As every action can only be used once before being forbidden, the EDD has a limited depth.

For each path from r to t in the EDD we can reconstruct a plan for Π by taking the actions chosen for application along the path. If we associate the cost of an action with the edge that applies it and a cost of 0 with the other edge, the cost of the path matches the cost of the plan. All plans without unnecessary actions are represented this way in one (applicable) permutation. Since an optimal plan has no unnecessary actions, the shortest path from r to t has cost h^+.

In order to turn this EDD into a RDD, we need to combine two nodes in a way that does not discard solutions over overestimate costs. Given two nodes u and v to be combined, we define the properties of the new node w as $F(w) = F(u) \cup F(v)$ and $N(w) = N(u) \cap N(v)$. With this definition, every path that can lead from u or v to t will also be applicable in w, so no solution is discarded. The cost of reaching w is the minimum over the cost of reaching u and v and the cost of reaching t from w cannot be higher than reaching it from u or v. Thus, no solution cost is overestimated and the RDD is guaranteed to overestimate the EDD. The cheapest path in the RDD is a lower bound to h^+ and can be used as an admissible heuristic in Π.

Note that our EDDs/RDDs are not ordered, i.e., several nodes on a path could branch over the same actions. However, applying an action adds its effects to the successor node and all its descendants so the action will never be applied twice. A branch can only mention an action a more than once if the first choice follows the successor where a is not applied. Even in those cases, a can only be selected again if a node was combined where a was not forbidden. As there are limited options for this, the algorithm terminates.

The method to construct the RDD can be used with different strategies for choosing which operator to branch over (operator selection strategy) and which nodes to combine (combination strategy). As an initial evaluation, we keep the operator selection strategy fixed and always select the first applicable operator according to an arbitrary order. We tested a combination strategy that selects a pair of nodes to combine that has minimal Hamming distance over its properties. To show its effect, we compare it to a baseline combination strategy that just selects two nodes at random.

Preliminary Experiments

We implemented the EDD and RDD construction described above in the Fast Downward planner (Helmert 2006). We tested two different values for the width ω, 128 and 256, and compared the random combination strategy to the one based on Hamming distance for all of them. We used a set of 921 IPC instances where we could compute h^+. On 171 of those, we could construct the EDDs. Each run had a limit of 3.6 GB for memory and 30 minutes for CPU time.

As expected, larger values of ω produce better approximations of h^+, but the process of selecting and combining nodes can become a bottleneck. When $\omega = 128$, using Hamming distances leads to better approximations than random node selection in 82% of the tasks. This rate increases to 95% when $\omega = 256$. However, neither combination strategy gets close to h^+ in larger instances because combining nodes without considering their cost introduces shortcuts.

Conclusion and Future Work

We presented a way to approximate delete-free planning tasks with RDDs using advantages of both DDs and delete-free planning semantics. However, initial results show that more effort is needed to make this idea competitive with the state-of-the-art delete-free heuristics for classical planning. In the future, we want to explore other ideas for operator selection and combination strategies. For instance, greedily selecting a landmark action whenever applicable may reduce the depth of some paths in the tree, while selecting nodes to combine also based on the cost from r to the nodes can minimize the risk of creating shortcuts.

References

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *Proc. CP 2007*, 118–132. Springer.

Bergman, D.; Cire, A. A.; Van Hoeve, W.-J.; and Hooker, J. 2016. *Decision diagrams for optimization*. Springer.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1-2):165–204.

Gefen, A., and Brafman, R. I. 2011. The minimal seed set problem. In *Proc. ICAPS 2011*.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: what's the difference anyway? In *Proc. ICAPS 2009*.

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