DTG-Plan:Fast Planning by Search in Domain Transition Graphs

Ruoyun Huang*, *Yixin Chen*, and *Weixiong Zhang
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, MO 63130, USA
{rh11,chen,zhang}@cse.wustl.edu

Abstract

Recent advances in classical planning have used the SAS+ formalism, and several effective heuristics have been developed based on the SAS+ formalism. Comparing to the traditional STRIPS/ADL formalism, SAS+ is capable of capturing vital information such as domain transition structures and causal dependencies. DTG-Plan uses a new SAS+ based incomplete planning approach. Instead of using SAS+ to derive heuristics within a heuristic search, we directly search in domain transition graphs (DTGs) and causal graphs (CGs) derived from the SAS+ formalism.

Introduction

The SAS+ formulation of planning problems has drown increasing attention recently in classical planning (Bäckström & Nebel 1995). In contrast to the traditional STRIPS formalism, the SAS+ formulation can provide compact constructs such as domain transition graphs (DTGs) and causal graphs (CGs) to capture vital domain information of domain transition structures and causal dependencies.

Many applications of the SAS+ formalism have been studied. In the Fast Downward planner (Helmert 2006), a heuristic function was developed by analyzing the causal graphs on top of the SAS+ models. Another SAS+ based heuristic for optimal planning was recently derived via a linear programming model encoding DTGs (van den Briel *et al.* 2007). Moreover, long-distance mutual exclusion (mutex) constraints based on a DTG analysis was proposed and shown to be effective in speeding up SAT-based optimal planners (Chen, Zhao, & Zhang 2007).

However, one limitation of the existing methods using the SAS+ formulation is that their high-level problem-solving strategy is searching for solution graphs or plans in STRIPS state spaces. In other words, the previous works focused mainly on deriving new heuristics or mutex constraints, while exploring a search space where each state is represented by binary STRIPS facts. It is critical to mention that a STRIPS state space can be very large for a typical planning problem. In particular, the number of binary facts (F) in a planning problem is typically in the order of 10^3 to 10^4, and the size of the state space is $\Omega(2^F)$, resulting in huge time and memory complexities in the worst case.

Although the compact constructs provided by the SAS+ formulation have been used to develop effective heuristics to speed up search (Helmert 2006; Chen, Zhao, & Zhang 2007), the domain transitions and causal dependencies encoded in SAS+ have not been fully exploited. DTG-Plan was motivated by the possibility of searching directly in graphs composed of DTGs which are further inter-connected by CGs. We are inspired by the observation that the compact constructs from the SAS+ formulation can give rise to small, compact graphs, which can be substantially smaller than a binary-fact state space. In particular, the size of any DTG for a given problem is often very small. The number of DTGs is typically in the order of 10 to 100, and the number of nodes in a DTG varies from a few to around 20 for a problem in the recent years' planning competitions. Therefore, the size of each DTG is small and a direct search on DTGs can be efficient.

On the other hand, the search of a plan cannot be completely decomposed into searching individual DTGs. DTGs can depend on one another due to causal dependencies, which lead to complex orderings among actions. Hence, causal dependencies are indeed the culprit of the difficulty of automated planning. One possible approach is to merge individual DTGs under the constraints of their causal dependencies, while maintaining the overall graph as small as possible so as to make the search process efficient. However, an effective DTG merging scheme seems to be difficult to come by. This certainly calls for a new approach to utilize DTGs and deal with their causal dependencies.

We developed DTG-Plan, which directly searches in a space of DTGs and CGs rather than a binary-fact search space. Based on the DTG structures, our algorithm directly extracts plans from a graph composed of DTGs. We distribute the decision making into several hierarchical levels of search to deal with causal dependencies. At each level, we design heuristics to order branching choices and prune non-promising alternatives. We show that the direct search of DTGs can work well across a variety of planning domains, showing competitive performance.

The method used in DTG-Plan has at least two advantages over the traditional heuristic search algorithms on STRIPS models. First, unlike the popular relaxed-plan heuristic that ignores delete effects, the DTGs preserve much structural information and help avoid deadends in problems from many

domains. Second, the proposed method introduces a hierarchy of decision-making where heuristics can be accordingly designed for different levels of granularity of search. In contrast to requiring a single good heuristic in planners such as FF (Hoffmann & Nebel 2001), the proposed method provides an extensible framework in which heuristics and control rules can be designed, incorporated and improved at multiple levels.

The main routine of DTG-Plan conducts search in an abstract space instead of the original problem space, while the other graphs are traversed in a hierarchical way. The ideas of decomposing are similar to that of previous hierarchical decomposition planning algorithms such as HTN planning. However, DTG-Plan is fully automated and does not require domain-specific control knowledge to specify how an action can be decomposed.

Background

In STRIPS planning, a **fact** f is an atomic fact that can be true or false. Given a set of facts $F = \{f_1, f_2, \ldots, f_n\}$, a STRIPS state S is a subset of facts in F that are set to true. An **action** a is a triplet a = (pre(a), add(a), del(a)), where pre(a)$\subseteq$ F is the set of preconditions of action a, and add(a)$\subseteq$ F and del(a)$\subseteq$ F are the sets of add facts and delete facts, respectively. A **planning task** is a triplet $(O, S^{initial}, S^{goal})$, where O is a set of actions, $S^{initial} \subseteq F$ the initial state, and $S^{goal} \subseteq F$ the goal state.

The SAS+ formalism (Helmert 2006; Jonsson & Bäckström 1998) of a planning domain includes a set of multi-valued **state variables**, where a variable represents a group of mutually exclusive facts from which only one can be true in any state. We denote the set of state variables as $X = (x_1, x_2, \ldots, x_m)$, where x_i takes a value from a finite discrete set $\mathcal{D}_i$. For a SAS+ task, the value assignment of a state variable corresponds to a binary fact in the traditional STRIPS formalism.

Definition 1 (Domain transition graph (DTG)) *Given a state variable $x \in X$ defined over $\mathcal{D}_x$, its DTG G_x is a directed graph with vertex set $\mathcal{D}_x$ and arc set $\mathcal{A}_x$. A directional arc (v, v') belongs to $\mathcal{A}_x$ if and only if there is an action o with $v \in del(o)$ and $v' \in add(o)$, in which case we say that there is a **transition** from v to v'. We use $T_{v,v'}$ to denote the set of actions that can **support** the transition from v to v': $T_{v,v'} = \{o | v \in del(o), v' \in add(o)\}$.*

For simplicity, we assume that each fact exists in only one DTG. A domain violating this assumption can be transformed into one satisfying it (Helmert 2006). Given a STRIPS fact f, if it corresponds to a state variable assignment in a DTG G, we denote it as $DTG(f) = G$, or simply $f \in G$. Given any two vertices $f, h \in G$, we define the **minimum DTG cost** $\Delta_G(f, h)$ as the distance of the shortest path from f to h in G.

There may exist **causal dependencies** between DTGs. Consider two DTGs G and G'. If an action o in G' has a precondition f in G, denoted as $DTG(f) = G$, we say G' depends on G. We denote $dep(G)$ as the set of DTGs depending on G.

Given a SAS+ planning domain with state variable set $X = (x_1, x_2, \ldots, x_m)$, each state variable x_i corresponds to a DTG G_{x_i}. A state corresponds to a complete value assignment to all the state variables. In the following, we use states to refer to SAS+ states. Given a state S, we use $\pi(G, S)$ to specify the fact that is true in the DTG G.

For a DTG G and two facts f and h in G, we define the **transition path set** $\mathcal{P}(G, f, h)$ to be the set of all possible paths from f to h in G, with the restriction that each vertex can appear at most once in a path.

The DTG Planning Algorithm

The task of extracting a plan from DTGs can be viewed as selecting a sequence of actions from the DTGs. At the top level, for each subgoal $g_i, i = 1..N$, we find their DTGs $DTG(g_i)$, then find a sequence of transitions in $DTG(g_i)$ that reaches g_i from the initial state. The transitions for various subgoals may need to be interleaved instead of sequentially concatenated. We need to find a proper way to interleave these transitions. For each selected transition, we need to select an action to materialize the transition. To satisfy the preconditions of the action, we may need to insert a sequence of necessary "supporting actions" before the action.

As described above, there are three major components in our search: 1) selecting a path of transitions between two nodes in a DTG (Procedure *search_paths()*), 2) deciding the execution order of facts (subgoals or preconditions) (Procedure *search_fact_set()*), and 3) selecting an action for a transition(Procedure *search_transition()*). To speedup the search, we use various heuristics to help choose promising directions and reduce the branching factors. Since we prune many branching choices at the three places above, our algorithm is an incomplete search.

Finding transition paths in the abstract space

The top-level procedure finds a high level plan of transitions for achieving all subgoals. It is denoted as search_goals(). We emphasize that the approach here is not incremental planning that concatenates sequentially plans for subgoals. Instead, the approach takes the interactions between different subgoals into consideration.

Procedure search_goals() works as follows. It behaves in a way similar to A$*$ search. For each unsatisfied goal g, we find its DTG $G = DTG(g)$, and generate the transition path set $\mathcal{P}(G, \pi(G, S), g)$, where $\pi(G, S)$ is the value of G in current state S. Rather than completing a path from S to g, we only try to move one transition further towards g for each step. The search_transition() procedure tries to find a concrete plan that realizes the transition $T_{u,v}$. All the applicable transitions (represented by a sequence of actions), along with the consequent states are evaluated by summing up the heuristic value and the length of the action sequences. That is to say, the states are evaluated by $f = g + h$. After that, we push the states into a queue. The next iteration will pick the first state in the queue and repeat the procedure specified above.

This procedure offers a complete search that can interleave the plans for subgoals for difficult problems where the

subgoal interactions need to be considered. Actually, once the searches in the goal level graphs terminates, no matter a solution is found or not, DTG-Plan can incorporate more graphs (i.e. not only the goal related ones) into the abstract space and start with a new search. Eventually it gets a solution with better quality.

A key observation is that, in almost all the IPC domains, the goal-level DTGs all have smaller size than other DTGs and the $\mathcal{P}(G, \pi(G, S), g)$ set often contains very few paths. In fact, many goal-level DTGs contain only one path to the subgoal and the paths are typically short. Therefore, the complexity of search_goals() tends to be low.

Materializing a transition

Procedure search_transition() takes a state S, a DTG G, a transition T, and a set of forced preconditions as parameters. This procedure chooses an action o to materialize the transition T. Before executing o, its preconditions and forced preconditions must also be true. Therefore, the procedure typically returns a plan consisting of a sequence of actions achieving the preconditions, followed by the action o at the end.

A transition usually has a number of supporting actions, from which one must be chosen. To choose one action, we sort all supporting actions by heuristic values. Given the transition T in a DTG G, we evaluate each of its supporting actions by estimating the number of steps needed to make all its preconditions true. Formally, given an action o and the current state S, we estimate the cost to enable o as the total minimum DTG cost of all preconditions of o:

$$cost(o) = \sum_{\forall f \in pre(o)} |\Delta_G(\pi(G, S), f)|.$$

In search_transition(), we sort all supporting actions in an ascending order of their costs. Actions with lower costs are tried earlier because their preconditions are likely to be easier to achieve.

Forced preconditions We denote the set of forced preconditions as forced_pre(T,p). When materializing a transition T in a path p in a plan, it is beneficial to look ahead at the full path p. A wrong plan for the transition may lead the search into a deadend. Although backtracking can solve this problem, it is more efficient to detect such situations beforehand and avoid deadends.

Given a path p of transitions in a DTG G, and a transition T on path p, we define the forced preconditions of T with respect to p as follows. Assume T turns a fact f to g, and then T_1, which is the next transition following T on path p, changes fact g to h. If there exists a fact q such that $q \in pre(o)$ for any action supporting T_1 and there is a forced ordering $q \prec g$ (Koehler & Hoffmann 2000), we call q a forced precondition of transition $T_{f,g}$. The set force_pre(T, p) includes all such forced preconditions of T.

The rationale for the above definition is the following. The fact q is required in order to execute T_1. We need to find a plan to satisfy q when we call search_transition() for T_1. However, in our case, it would be too late due to the $q \prec g$ ordering, which means that g has to be invalidated before achieving q. Thus, we need to achieve q *before* g. In search_transition(), a forced precondition, such as q, will be treated as a precondition for transition T.

Search for a valid transition path

Given a state S, a graph G and two facts $f_1, f_2 \in G$, procedure search_paths() tries to find a valid transition path from f_1 to f_2. First, we compute the set of possible paths $\mathcal{P}(G, f_1, f_2)$. For each p in $\mathcal{P}(G, f_1, f_2)$, we first compute its forced preconditions and then traverse along p to form a plan (Lines 6-11). For each transition T in p, we call procedure search_transition() to search for a plan. A plan is returned when solutions to all transitions of a path p is found.

Search for a set of preconditions

Given a state S and a set of facts F, starting at S, the procedure search_fact_set() searches for a valid plan that can make all facts $f \in F$ be true at a final state. The facts in F may require a particular order to be made true one by one.

For the facts in F, we derive the following partial orders between them, listed with a descending priority.

O1. For two facts $f, h \in F$, if $DTG(h) \in dep(DTG(f))$, we order f before h. To understand the reason, consider a transportation domain with a cargo and a truck. The DTG of a cargo depends on that of a truck. If we fix the truck at a location, then we may not be able to move the cargo without moving the truck. Thus, it makes sense to first deliver the cargo before relocating the truck.

O2. For each fact $f \in F$, we evaluate a degree-of-dependency function defined as

$$\beta(f) = |dep(DTG(f))| - |dep^{-1}(DTG(f))|,$$

where $dep^{-1}(G)$ is the set of graphs that G depends on. For two facts $f, h \in F$, if $\beta(f) > dep(DTG(h))$, we order f before h. This is a generalization of O1.

O3. For two facts $f, h \in F$, if there is a forced ordering $f \prec h$, we order f before h.

In search_fact_set(), we first consider all the permutations that honor the above partial orderings. In very rare cases, when all permutations that meet O1, O2 and O3 fail, we continue to compute the remaining orderings.

In some domains, an action may have a large number of preconditions that share the same predicates. Given two facts f and g, if f and g have the same predicate, and they differ by only one object arity, we call f and g **symmetric facts.** Given an action o, if there exist a number of facts $f_1, f_2, \ldots, f_n$, $f_i \in pre(o)$, such that every two f_i and f_j are symmetric facts, we ignore the differences between all of these facts when generating orderings. Based on the partial-order heuristic and symmetric-fact pruning, we have a good chance to succeed quickly. We usually get a feasible ordering among the first three candidates.

Conclusions

We presented DTG-Plan, which is for classical planning based on the SAS+ formalism. The algorithm directly

searches for plans in a graph composed of DTGs. We distributed the decision making over several search hierarchies to deal with causal dependencies. For each level, we developed heuristics to order branching choices and to prune nonpromising alternatives.

The hierarchy of the proposed search algorithm used in DTG-Plan may seem to be complex, but very often the search can extract a plan quickly along a depth-first path with little backtracking. Since the sizes of DTGs are generally small, the search can be very efficient with proper pruning and ordering heuristics.

References

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:17–29.

Chen, Y.; Zhao, X.; and Zhang, W. 2007. Long distance mutual exclusion for propositional planning. In *Proceeding of International Joint Conference on Artificial Intelligence*, 1840–1845.

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

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Jonsson, P., and Bäckström, C. 1998. State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence* 100(1-2):125–176.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research* 12:338–386.

van den Briel, M.; Benton, J.; Kambhampati, S.; and Vossen, T. 2007. An LP-Based Heuristic for Optimal Planning. In *Proc. Thirteenth Intel. Conference on Principles and Practice of Constraint Programming*.