Speeding up Planning through Minimal Generalizations of Partially Ordered Plans

Radomír Černoch and Filip Železný

Czech Technical University in Prague
{cernorad,zelezny}@fel.cvut.cz

Abstract. We present a novel strategy enabling to exploit existing plans in solving new similar planning tasks by finding a common generalized core of the existing plans. For this purpose we develop an operator yielding a minimal joint generalization of two partially ordered plans. In three planning domains we show a substantial speed-up of planning achieved when the planner starts its search space exploration from the learned common generalized core, rather than from scratch.

1 Introduction

Automated planning has been a core area in artificial intelligence research for decades [1]. Nevertheless, novel planning algorithms are still being designed and several international annual competitions demonstrate the perpetual improvement in their performance. Of course, this unceasing progress also indicates the persisting room for improvement. To this end, a lively research direction aims at improving a planner by exploiting experience obtained in previously completed planning tasks. This general strategy has a clear motivation in that a typical deployment of a planner is in repetitive planning tasks only slightly varying in their initial conditions and desired goals. The specific approaches to exploiting previous plans differ mainly in two respects: what kind of knowledge they extract from the old plans, and what techniques they employ for that extraction.

As for the former aspect, a popular state-of-the-art approach is to look for structures (e.g. sequences) of actions that are frequently found in the plans [2–4]. Such actions are then glued into a single *macro-operator* made available to the planner. In other approaches [5, 6], *control rules* are extracted serving as heuristics for the choice of a suitable action given the preceeding actions already in the plan.

As for the latter aspect, techniques for analyzing previous plans may roughly be projected onto a spectrum between the 'deductive' and 'inductive' extremes. An example of the deductive approaches is [4] which determines the dependence structure in plans and suggests action sets which can act as macro-operators. Halfway in the spectrum are algorithms which search frequent action-set patterns in a narrowly prescribed form [3]. Lastly, established machine learning algorithms have also been explored in this context, such as for learning Markov chains for probabilistic description of action sequences [6].

Here we are mainly interested in learning from plans using the expressive formalism of first-order logic. This is a natural choice since planning problem descriptions and plans themselves are typically encoded in fragments of first-order logic. Previous research in this direction explored the application of explanation based learning [7]. Also ILP applications in the planning area have been reported, dating back to early works on relational reinforcement learning [8]. ILP has also been employed for the already mentioned standard task of learning control rules [9, 10].

By commonsense assessment of all the above reviewed approaches, we think they share one important deficiency. If we apply a planning algorithm repeatedly with only slightly varying task descriptions, we may expect the resulting plans to also vary only slightly. In other words, they may share a very large core structure, perhaps even containing the majority of actions inside the plans. Under such circumstances, it is clearly underambitious to use old plans only for seeking small nuggets such as control rules or search heuristics. Indeed, it is more adequate to directly determine the entire common core of the plans rather than just learn heuristics to guide their reconstruction. Finding such largest common cores is the goal pursued by the current study. The way such findings will be exploited by the planner is also different from the current approaches. Rather than producing new macro-operators or control rules, the discovered core plan structure will directly be used as an initial incomplete plan to be refined by the planner towards satisfying all initial conditions and goals of the current task. In other words, rather than equipping the planner with new operators or heuristics, we will advise a specific place in its search space where to begin the plan refinement.

Finding shared plan cores does not simply correspond to detecting the largest set of actions found in all input plans. Roughly, there are two factors making the problem more complicated and interesting. First, we must respect action dependencies reflected by a partial order defined on each input plan. Second, in learning a plan core, we should be able to abstract e.g. from domain-specific object names by means of variables. The presence of a candidate structure in a plan will thus be checked in a way more resembling θ-subsumption than the subset relation.

In terms of description complexity, plan cores will obviously be larger than learned control rules or heuristic functions. It is unrealistic to expect that *top-down* (general-to-specific) learning approaches would scale to searching among candidate structures possibly containing tens or hundreds or more actions. We therefore base our approach on a *bottom-up* strategy where input plans are jointly generalized. The central operator we develop and employ for this purpose is one that produces a minimal joint generalization of two partially ordered plans. The operator is obviously inspired by Plotkin's least general generalization [11] of clauses. Unlike operators previously developed for totally ordered clauses [12, 13], here we work with partial orders.

The paper is organized as follows. In the next section we explain the method for the joint generalization of partially ordered plans. In Section 3 we evaluate the method empirically and in Section 4 we conclude the paper.

2 Method

A plan is simply a partially ordered set of actions, where an action is a first-order atom such as $\mathsf{pickup}(\mathsf{block}, \mathsf{lefthand})$.

Definition 1. *A plan $P = (A, \leq)$ consists of a set of atoms $A = \{a_1, ..., a_n\}$ and a reflexive, transitive and antisymmetric relation $\leq$ on A.*

Note that terms in actions need not be ground. In fact, in what follows we specifically aim at generalizing plans and in so doing, we will turn constants into variables. Such generalized non-ground plans will act as a starting point for further refinements conducted by a planner in a new learning task, and variables will be grounded only as a result of these refinements.

Further in the text we will denote the transitive reduction of $\leq$ as $\leq^0$. Using the antisymmetry of $\leq$, the transitive reduction $\leq^0$ is always unique. We now proceed to defining a generality order on plans using the well known concept of OI-substitution [14], in which the variable-term mapping is injective.

Definition 2. *Let $K = (A_K, \leq_K)$, $L = (A_L, \leq_L)$ be plans. Plan K subsumes plan L iff there is an object-identity (OI) substitution θ such that*

$$A_K\theta \subseteq A_L \text{ and } \leq^0_K \theta \subseteq \leq^0_L \ .$$

We will denote the subsumption relation as $K \rhd_\theta L$.

Example 1. Consider the following plans:

$$L = (\{m = \mathsf{pick}(\mathsf{box}), n = \mathsf{move}(\mathsf{a,b}), o = \mathsf{drop}(\mathsf{box})\}, \{m \leq n, n \leq o\})$$
$$K_1 = (\{p = \mathsf{pick}(Z), q = \mathsf{move}(\mathsf{a,b})\},\ \{p \leq q\})$$
$$K_2 = (\{r = \mathsf{pick}(X), s = \mathsf{drop}(Y)\},\ \{r \leq s\})$$

Obviously we can see that $K_1 \rhd_\theta L$, where $\theta = \{Z \backslash \mathsf{box}\}$. Then note that $K_2 \not\rhd_\theta L$, because subsumption is defined through the transitive reduction $\leq^0$ rather than the partial order $\leq$ itself.

The reason for relying on OI-substitution as opposed to its standard counterpart is straightforward. If we had used standard substitution, several actions in a generalized plan could correspond to a single action in the more special (training) plans. This would be counter-intuitive: if all training plans contain a pick action only once, we do not want the generalized plans to contain this action multiple times (e.g., contain both $\mathsf{pick}(\mathsf{box}, X, \mathsf{robot})$) and $\mathsf{pick}(\mathsf{box}, \mathsf{room}, Y)$).

Example 1 emphasizes the fact that the definition of $\rhd$ subsumption uses the transitive reduction $\leq^0$ rather than the transitive relation $\leq$ itself. The reason is that generalized plans should not contain two preceding actions regardless of what happens in-between. Instead we seek sets of actions, which are executed in a block without being interleaved by any other actions.

Using the generality order we now define common generalization of plans.

Definition 3. *Let G, K, L be plans. Plan G is a* generalization *of K and L ($G = K \diamond L$) iff $G \rhd_\theta K$, $G \rhd_{\theta'} L$ and the order of generalized actions is preserved:*

$$\forall a, b \in A_G.\ (a \leq_G b) \rightarrow (a\,\theta \leq_K b\,\theta) \wedge (a\,\theta' \leq_L b\,\theta')$$

The generalization G of K and L is called a minimal generalization iff $G = K \diamond L$ and there is no G' such that $G \neq G'$, $G \rhd G'$ and $G' = K \diamond L$.

Example 2. In general, there can be multiple minimal generalizations of two plans. Consider the plans

$$\begin{aligned} K &= (\{\mathsf{P(a,b)}, \mathsf{P(c,d)}\},\quad \mathsf{P(a,b)} \leq \mathsf{P(c,d)}) \\ L &= (\{\mathsf{P(a,d)}\}, \emptyset) \end{aligned}$$

for which we want to find minimal generalizations. If $\mathsf{P(a,d)}$ from L is identified with $\mathsf{P(c,d)}$ from K, their minimal generalization is $G_1 = \mathsf{P}(X, \mathsf{d})$. Alternatively we can identify $\mathsf{P(a, d)}$ with $\mathsf{P(a, b)}$, which gives a minimal generalization $G_2 = \mathsf{P}(\mathsf{a}, Y)$. Note that neither $G_1 \rhd G_2$ nor $G_2 \rhd G_1$ and no more specific generalizations can be found. Hence both G_1 and G_2 are minimal generalizations of K and L.

This process described above will give rise to the the definition of least generalization with respect to a given mapping[1] between atoms of generalized plans.

Definition 4. *Let K, L, G be plans. Let there be an injective partial function $f : A_K \rightarrow A_L$. Then we say that G is the least generalization of K and L with respect to f iff $G = K \diamond L$, $G \rhd_\theta K$ and $G \rhd_{\theta'} L$, where θ and θ' are obtained from the anti-unification algorithm on pairs $(k, f(k))$, where $k \in \mathrm{domain}(f)$.*

The definition of least generalization with respect to a mapping is practical for finding minimal generalizations. It reduces the task to finding a correspondence between atoms in plans, because the θ substitutions are obtained from the deterministic anti-unification algorithm. Relying on the above introduced concepts, we designed an algorithm for finding minimal generalizations of 2 plans (K and L):

1. The essential phase constructs the mapping $f : A_K \rightarrow A_L$ with a *depth-first search* approach. Initially f is empty. A pair of actions from K and L is added to f if they share the predicate symbol (which represents the action name). Then the relations $\leq'_{\{K,L\}}$ are constructed as subsets of $\leq_{\{K,L\}}$ on all atoms present in $A_{\{K,L\}}$.
   If these relations satisfy the conditions in Definitions 2 and 3, the algorithm iterates with the newly constructed f. Otherwise the new f is discarded and the algorithm backtracks.
   The first phase ends if it is not possible to add any more actions into f.
2. The atoms A_G in the generalized plan G are obtained from the *anti-unification* algorithm on pairs of atoms in f.
3. The relation $\leq_G$ is taken directly from $\leq'_K$ and $\leq'_L$. Note the given the conditions in Definitions 2 and 3, both relations should have the same structure.

[1] The mapping corresponds to the concept of *selection* used in ILP literature.

3 Experiments

Here we test whether our plan-generalization approach allows to reduce planning times when compared to two baseline planning regimes.

Materials. In all experiments, we employ the *Plan-space algorithm* [1] as a representative of planning algorithms using the classical *STRIPS* representation based on first-order logic. Plan-space is a backward-chaining planning algorithm with the "least-commitment strategy": 2 actions are not ordered unless they are required to. Hence the produced plan is a partially-ordered set of actions unlike the case of state-space planning.

We have used 3 different planning domains (gold-miner, rover, and gripper) each containing a single "etalon" problem. A *problem* is a tuple (I, G) where I is the set of initial conditions and G is the set of goals. The etalon problem for domain d is denoted (I_d, G_d). As further experimental material, we use random planning problems (I, G) generated from the etalon problems in such a way that $I = I_d$ while G contains goals randomly sampled from G_d. Given that the etalon problems are solvable, it follows that these randomly generated problems are also solvable.

Domain	Initial propositions	Goal propositions	Plan length	Time to solve
`gripper`	9	7	9	7m
`goldminer`	26	5	9	42m
`rover`	18	8	8	1h 15m

Table 1: Properties of the *etalon problem* in all domains.

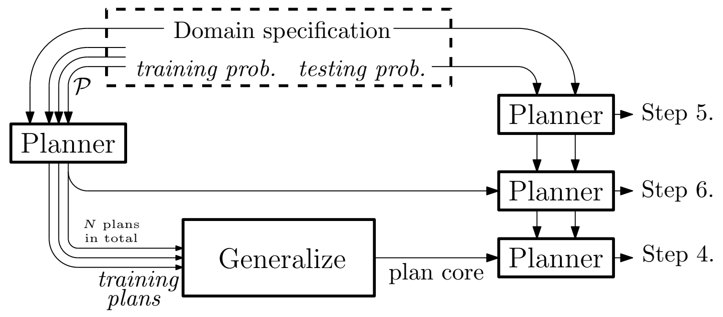

Fig. 1: Workflow of the testing process.

Protocol. We tested our approach using the following workflow (refer to Fig. 1) reflecting its anticipated use. For each domain d, each $S \in \{20\%, 40\%, 60\%, 80\%\}$ and each $N \in \{3, 4, 5\}$:

1. We generate the set $\mathcal{P}$ of all planning problems (I_d, G) such that $G \subseteq G_d$ and $\frac{|G|}{|G_d|} \doteq S$. The latter condition states that the goal sets of the two problems are similar to the degree of S.

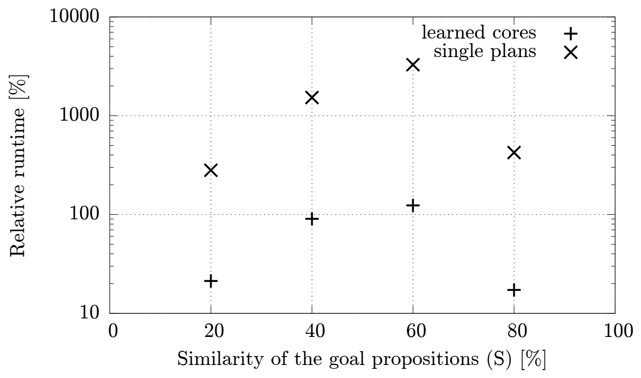

Fig. 2: Relative runtimes corresponding to Steps 4 (‘learned cores’) and 6 (‘single plans’) of the experimental protocol. The runtime for Step 5 of the protocol, i.e. for planing from scratch, corresponds to 100% in the figure.

S	From scratch	Plan-core	S	From scratch	Plan-core
20%	209%	499%	60%	441%	656%
40%	486%	498%	80%	158%	630%

Table 2: Effects of plan cores on the variance of runtimes. The values are standard deviations divided by the mean of the runtimes from Steps 5 and 4 of the protocol, factorized over S.

2. We randomly pick N problems from $\mathcal{P}$ to act as the *training problems*. For each of these problems, the planner generates its optimal plan. These plans constitute the *training set*. We further pick another, *testing problem* from $\mathcal{P}$.
3. *Plan cores* are obtained by generalizing the training plans according to the method described in Section 2. Since the method produces multiple possible joint generalizations, we randomly pick 30 distinct plan cores from the resulting set for evaluation.
4. For each of the 30 plan cores, the planner is launched on the testing problem, starting from a partially constructed plan corresponding to the plan core, and runtime needed to generate a plan is measured, and then averaged over the 30 plan cores.
5. As in Step 4 but the planner starts from scratch rather than from the learned plan core.
6. As in Step 4 but the planner starts from a partially constructed plan corresponding to a single plan randomly selected from the training plans, rather than from the learned plan core.

Considering Step 1, simple statistical reasoning yields that the smaller the value of S is, the less the training problems generated in Step 2 will be mutually similar, and the less the testing problem will be similar to the training problems.

By comparing results from Steps 4 and 5, we shall assess the influence of our approach to exploiting existing plans, against the situation where existing

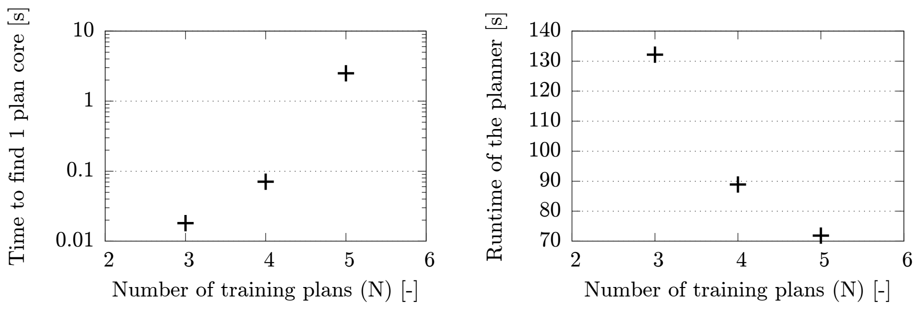

Fig. 3: Average runtimes for learning (top) and planning (bottom) as a function of the number of training plans.

plans are completely ignored. However, comparing results of Steps 4 and 6 is also important, since Step 6 represents a trivialized variant of our approach, in the sense that it does not require joint generalization of training plans. By this comparison, we thus evaluate the added value brought by the extra theory developed in Section 2.

Results. Fig. 2 shows the resulting runtimes factorized by the 3 strategies corresponding to Steps 4, 5, 6 and the value of S, and averaged over the remaining parameters d and N. Planning procedures not terminating in 1 hour were curtailed and assigned the 1 hour runtime value.

Despite the small slowdown for $S = 60\%$ and the increased variance in runtime, the overall results show that plan cores improved the efficiency of the planner. The overall average result with plan cores achieved 39% runtime of the original. Importantly, results for the scenario where the planner used single training plans instead of generalized plan cores show approximately $10\times$ increase in runtime, which justifies the usage of the learning algorithm.

Of relevance, the absolute time needed for learning is rather negligible in comparison to planning runtimes. This follows from Fig. 3 which plots the runtimes for learning and runtimes for planning using generalized plan cores, factorized by N (i.e., the number of used training plans) and averaged over S and d. Expectedly, the learning times grow and the planning times fall as N increases and its optimal value is, in general, a matter of trade-off.

4 Conclusions

We presented a novel strategy enabling to exploit existing (training) plans in new similar planning tasks by finding a common generalized core of the training plans. In experiments, we showed a substantial speedup of planning resulting from using this strategy. We tested our method in the envisioned application scenario where new planning tasks differ only slightly from the completed tasks used for learning. In particular we assumed that the old and new tasks share the same initial conditions and their goal sets are randomly sampled from the same base. In future work, it would be interesting to refine and test our method also in different instantiations of the similarity assumption, e.g. by allowing that the initial conditions of the new tasks may be different from those of the completed tasks.

Acknowledgment

We gratefully acknowledge the financial support of the Czech Science Foundation (GA ČR) for the years 2008-2010 (grant no. 201/08/0509).

References

1. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: theory and practice. Morgan Kaufmann Publishers (2004)
2. Botea, A., Enzenberger, M., Mller, M., Schaeffer, J.: Macro-FF: improving AI planning with automatically learned macro-operators. Journal of Artificial Intelligence Research **24** (2005) 581621
3. Coles, A., Smith, A.: Marvin: a heuristic search planner with online macro-action learning. Journal of Artificial Intelligence Research **28** (2007) 119–156
4. Chrpa, L., Bartak, R.: Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In: Florida Artificial Intelligence Conference. (2008)
5. Fernandez, S., Aler, R., Borrajo, D.: Using previous experience for learning planning control knowledge. In: Florida Artificial Intelligence Conference. (2004)
6. Yoon, S.: Learning heuristic functions from relaxed plans. In: International Conference on Automated Planning and Scheduling, AAAI Press (2006)
7. Kambhampati, S., Yoon, S.: Explanation based learning for planning. In: Encyclopedia of Machine Learning. Springer (2010)
8. Dzeroski, S., Raedt, L.D., Blockeel, H.: Relational reinforcement learning. In: Machine Learning. (1998) 7–52
9. Huang, Y., Selman, B., Kautz, H.: Learning declarative control rules for constraint-based planning. In: International Conference on Machine Learning. (2000)
10. Lorenzo, D., Otero, R.P.: Learning logic programs for action-selection in planning. In: Third International Workshop on Extraction of Knowledge from Databases at the 10th Portuguese Conference on Artificial Intelligence. (2001)
11. Plotkin, G., Meltzer, B., Michie, D.: A note on inductive generalization. Machine Intelligence **5** (1970) 153–163
12. Kuwabara, M., Ogawa, T., Hirata, K., Harao, M.: On generalization and subsumption for ordered clauses. In: New Frontiers in Artificial Intelligence. (2006)
13. Tamaddoni-Nezhad, A., Muggleton, S.: The lattice structure and refinement operators for the hypothesis space bounded by a bottom clause. Machine Learning **76** (2009) 37–72
14. de Raedt, L.: Logical and relational learning. Springer (2008)