Optimizing Plans through Analysis of Action Dependencies and Independencies

Lukáš Chrpa, Thomas Leo McCluskey and Hugh Osborne
Knowledge Engineering and Intelligent Interfaces Research Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, t.l.mccluskey, h.r.osborne}@hud.ac.uk

Abstract

The problem of automated planning is known to be intractable in general. Moreover, it has been proven that in some cases finding an optimal solution is much harder than finding any solution. Existing techniques have to compromise between speed of the planning process and quality of solutions. For example, techniques based on greedy search often are able to obtain solutions quickly, but the quality of the solutions is usually low. Similarly, adding macro-operators to planning domains often enables planning speed-up, but solution sequences are typically longer. In this paper, we propose a method for optimizing plans with respect to their length, by post-planning analysis. The method is based on analyzing action dependencies and independencies by which we are able to identify redundant actions or non-optimal sub-plans. To evaluate the process we provide preliminary empirical evidence using benchmark domains.

Introduction

Automated planning (Ghallab, Nau, and Traverso 2004) even in its simplest form, classical planning, is intractable (PSPACE-complete) (Bylander 1994). Despite the general complexity results there are many classes of planning problems that are tractable (i.e., solvable in polynomial time) (Helmert 2003; 2006) but sometimes finding an optimal solution (in classical planning, the shortest solution) is intractable (NP-hard). Fink and Yang (1992) showed that plan optimization is NP-complete in general.

Existing planners must compromise between speed of the planning process and quality of solutions. In some applications it is necessary to provide any solution as fast as possible to avoid imminent danger or damage to a physical agent. A good example of fast but non-optimal planning techniques is performing greedy search such as in LPG (Gerevini, Saetti, and Serina 2004). Moreover, if macro-operators are involved, then optimality might be affected even though we use an optimal planner.

Approaches to plan optimization using genetic programming are promising, though the relation between plan generation time and optimization time is unclear (Westerberg and Levine 2001). The most recent related work (Nakhost and Müller 2010) proposes two approaches for plan optimization. First, it tries to remove a pair of (dependent) actions. If the rest of the plan remains valid then an action can be removed. Second, it expands a limited number of nodes around each state along the plan to a produce a Neighborhood Graph and then, by applying Dijsktra's algorithm, it finds a shortest path within the neighborhood. This method is restricted to local optimality, however, and does not exploit the information within the plan structure (e.g some actions might lie far from each other in a plan but can be adjacent in some permutation of the plan).

Redundant actions in a plan are those that do not contribute to the rest of the plan, or are inverse actions reversing each other's effects. These inverse actions do not have to be necessarily adjacent in (sequential) plans. In this paper, we introduce a method which exploits plan structure, drawing on Chrpa's work on analysis of action dependencies and independencies (Chrpa 2010), to identify such redundant actions. Moreover, the method can detect a pair of actions (also not necessarily adjacent in plans) that can be replaced by a single action. We evaluate the method analytically, by considering time complexity bounds, and empirically, using benchmark domains. We postulate some promising directions for future work, for instance to investigate whether our method (or an extension) could be used for optimal planning with macro-operators.

Preliminaries

Classical planning (in state space) deals with finding a sequence of actions transforming the environment from some initial state to a desired goal state. The environment is static, deterministic and fully observable.

In the set-theoretic representation **atoms**, which describe the environment, are propositions. **States** are defined as sets of propositions. **Actions** are specified via sets of atoms specifying their preconditions, negative and positive effects (e.g., $a = \{pre(a), eff^-(a), eff^+(a)\}$). An action a is **applicable** in a state s iff $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$.

In the classical representation atoms are predicates. **Planning operators** are generalized actions, where preconditions, negative and positive effects are sets of (unground) predicates. The set-theoretic representation can be obtained from the classical representation by grounding.

A **planning domain** is specified via sets of predicates and planning operators (alternatively propositions and actions). A **planning problem** is specified via a planning domain, initial state and set of goal atoms. A **plan** is a sequence of actions. A plan is a **solution** of a planning problem if and only if a consecutive application of the actions in the plan (starting in the initial state) results in a state, where all the goal atoms are satisfied.

Action Dependencies and Independencies

Naturally, actions influence each other. An action provides atoms which are preconditions for some actions but on the other hand 'clobber' atoms required by other actions (Chapman 1987). Having a sequence of actions we can identify dependencies between the actions in terms of which actions provide atoms to other actions that need them as their precondition. The formal definition follows.

Definition 1. *Let $\langle a_1, \ldots, a_n \rangle$ be an ordered sequence of actions. An action a_j is* ***directly dependent*** *on an action a_i (denoted as $a_i \rightarrow a_j$) if and only if $i < j$, $(\mathit{eff}^+(a_i) \cap \mathit{pre}(a_j)) \neq \emptyset$ and $(\mathit{eff}^+(a_i) \cap \mathit{pre}(a_j)) \not\subseteq \bigcup_{t=i+1}^{j-1} \mathit{eff}^+(a_t)$.*
An action a_j is ***dependent*** *on an action a_i if and only if $a_i \rightarrow^* a_j$ where $\rightarrow^*$ is the reflexive transitive closure of the relation $\rightarrow$.*
$\nrightarrow$ denotes that actions are not directly dependent and $\nrightarrow^*$ denotes that actions are not dependent. ■

The notion of action independence is motivated by the possibility of swapping adjacent (independent) actions without breaking plan validity (proven in (Chrpa and Barták 2008)). Note that action independence as known in parallel planning (Blum and Furst 1997; Rintanen, Heljanko, and Niemelä 2006) has slightly different meaning, i.e., it does not depend on action ordering while in our case it does.

Actions a_i and a_j are independent (can be swapped if adjacent) if a_j is not dependent on a_i, a_j must not 'clobber' atoms from precondition of a_i and a_i must not 'clobber' any positive effect of a_j. Therefore, action independence is not the complement of action dependence.

Definition 2. *Let $\langle a_1, \ldots, a_n \rangle$ be an ordered sequence of actions. Actions a_i and a_j (without loss of generality we assume that $i < j$) are* ***independent*** *(denoted as $a_i \leftrightsquigarrow a_j$) if and only if $a_i \nrightarrow^* a_j$, $\mathit{pre}(a_i) \cap \mathit{eff}^-(a_j) = \emptyset$ and $\mathit{eff}^+(a_j) \cap \mathit{eff}^-(a_i) = \emptyset$.*
$\not\leftrightsquigarrow$ denotes that actions are not independent (but not necessarily dependent). ■

To obtain a complete model of these relations in plans (solutions of planning problems) we have to use two special actions: $a_0 = \{\emptyset, \emptyset, I\}$ (I is an initial state) and $a_g = \{G, \emptyset, \emptyset\}$ (G is a set of goal atoms). Using the main property of action independence we can shuffle actions in plans without affecting their validity (for more insight regarding permutation on plans, see (Fox and Long 1999)). This can be useful for checking whether actions can be adjacent which has been used for generating macro-operators (Chrpa 2010).

The check whether actions a_i and a_j can be adjacent is achieved by iteration of any of the following steps, which will attempt to reposition intermediate actions so that they are no longer between a_i and a_j.

1. If $a_i \leftrightsquigarrow a_{i+1}$, then move a_{i+1} before a_i.
2. If $a_{j-1} \leftrightsquigarrow a_j$, then move a_{j-1} after a_j.
3. Let $X = \{x \mid a_i \not\leftrightsquigarrow a_x \wedge i < x < j\}$ be a set of indices. If $X \neq \emptyset$ select k as the largest element of X and if $\forall l \in \{k+1, \ldots, j\} : a_k \leftrightsquigarrow a_l$, then move a_k after a_j
4. Let $X = \{x \mid a_x \not\leftrightsquigarrow a_j \wedge i < x < j\}$ be a set of indices. If $X \neq \emptyset$ select k as the smallest element of X and if $\forall l \in \{i, \ldots, k-1\} : a_l \leftrightsquigarrow a_k$, then move a_k before a_i

If a_i and a_j become adjacent (all intermediate actions have been successfully repositioned), then we say that a_i and a_j are **weakly adjacent**.

Relations of action dependence and independence can be computed in $O(n^2)$ steps (n is a length of the plan) and deciding weak adjacency can be done in at worst $O(l^2)$ steps (l is the number of intermediate actions) (Chrpa 2010).

Optimizing Plans

Using the theoretical framework above we can identify some redundant actions in plans. For instance, if the goal action a_g is not dependent on an action a_i, then a_i is redundant because it does not provide any atoms to actions (including a_g) that somehow support the goal. The formal proof follows.

Proposition 1. *Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan solving planning problem P and $a_g = \{G, \emptyset, \emptyset\}$ (G is a set of goal atoms in P) be an action. Let $A^- = \{a_i \mid a_i \in \pi, a_i \nrightarrow^* a_g\}$ be a set of actions on which the goal is not dependent. Then $\pi' = \pi \setminus A^-$ is also a solution of P.*

Proof. Let $A^0 = \{a_i \mid a_i \in \pi, \forall j : a_j \in \pi \cup \{a_g\}, a_i \nrightarrow a_j\}$ be a set of actions such that no other action is directly dependent on them. Obviously $A^0 \subseteq A^-$. According to Definition 1 we know that no action from π and the goal action a_g requires atoms provided by actions from A^0 thus actions from A^0 can be removed from π. Let $A^1 = \{a_i \mid a_i \in \pi, \forall j : a_j \in (\pi \cup \{a_g\}) \setminus A^0, a_i \nrightarrow a_j\}$ be a set of actions such that no other action from $\pi \setminus A^0$ and a_g is directly dependent on them. Obviously $A^1 \subseteq A^-$. If actions from A^0 are removed from π, then there is no longer an action (directly) dependent on actions from A^1. Using the same principle we can remove A^1 from π. We can construct $A^2, \ldots A^k$ analogously until $A^{k+1} = \emptyset$ and show that actions in these sets can be removed from π. It can be easily observed from the acyclicity of the relation of direct dependency that $A^- = A^0 \cup A^1 \cup \cdots \cup A^k$ and $k \leq n$). □

We say that action a_j is **inverse** to a_i if for every state s a consecutive application of a_i and a_j results back in s or a_i is not applicable in s. Inverse actions, i.e. actions that reverse effects of each other, are also redundant if they are executed successively. These actions might not be necessarily adjacent in plans but still redundant. If weakly adjacent actions are inverse, then they can be removed. However, checking weak adjacency might be quite expensive (at worst case $O(n^2)$, see (Chrpa 2010)). In the following proposition we show that if a_j is inverse to a_i and there is no action placed between a_i and a_j such that it is directly dependent on a_i or 'clobbers' any atom given back by a_j, then a_i and a_j can be removed.

Proposition 2. *Let $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan solving planning problem P. Let $a_i, a_j \in \pi, i < j$ be actions such that a_j is inverse to a_i. If there is no action a_k $(i < k < j)$ such that $a_i \rightarrow a_k$ or $\mathit{eff}^-(a_k) \cap \mathit{eff}^+(a_j) \neq \emptyset$, then $\pi' = \pi \setminus \{a_i, a_j\}$ is also a solution of P.*

Proof. From the assumption we know that a_j deletes all positive effects produced by a_i, therefore a_i is not essential for actions applied after a_j. If there exists a_k $(i < k < j)$ such that $a_i \rightarrow a_k$, then according to Definition 1 removing a_i causes inapplicability of a_k thus the plan will no longer be valid. Otherwise removing a_i from π will not affect the applicability of the other actions (except a_j which is removed as well). If there exists a_k $(i < k < j)$ such that $\mathit{eff}^-(a_k) \cap \mathit{eff}^+(a_j) \neq \emptyset$, then removing a_j might cause inapplicability of actions after a_j. From the definition of inverse actions we know that all atoms from $\mathit{eff}^+(a_j)$ are present in the state before a_i is applied. If no such a_k removes any of these atoms, then we do not need a_j to produce them again and a_j (together with a_i) can be removed from π. □

Using the above proposition reduces the worst case complexity of detecting redundancy of a pair of inverse actions in plans to $O(n)$ (if the number of intermediate actions is close to the plan length). For all possible pairs it is $O(n^3)$.

Besides detecting redundant inverse actions in plans we can also identify a pair of weakly adjacent actions that can be replaced by a single action. In (Chrpa 2010) weakly adjacent actions are used for generating macro-actions. Here we do not need macro-actions but single actions which can be used instead of pairs of actions used in plans. We say that an action a is **replaceable** by an action a' if $\mathit{pre}(a') \subseteq \mathit{pre}(a), \mathit{eff}^-(a') \subseteq \mathit{eff}^-(a)$ and $\mathit{eff}^+(a') \supseteq \mathit{eff}^+(a)$. Therefore our approach differs from (Chrpa 2010) in such a way that instead of introducing a new macro-action which can be used instead of a pair of weakly adjacent actions we look for a single action such that the (potential) new macro-action is replaceable by the (single) action. If so that we use the (single) action instead of the pair of weakly adjacent actions. It shortens the plan by one. The worst case complexity (for one pair of actions) is $O(n^2)$ (if the number of intermediate actions is close to the plan length). For all possible pairs it is $O(n^4)$.

Plans can be optimized in the following steps:

1. Compute action dependencies. Remove all actions on which the goal is not dependent (see Proposition 1).
2. Compute action dependencies. Identify and remove all pairs of inverse actions if they follows Proposition 2. Repeat until no action is removed.
3. Compute action dependencies and independencies. Identify pairs of weakly adjacent actions which can be replaced by a single action (and replace if applicable). Continue with step 2 if any pair is replaced.

Note that repetition of steps 2 and 3 might be necessary for identifying further non-optimalities. For instance, we can reveal 'nested' inverse actions, i.e., if we have a sequence a_i, a_p, a_q, a_j, where a_j is inverse to a_i and a_q is inverse to a_p, then if a_p (or a_q) is directly dependent on a_i, then a_i and a_j cannot be removed at this stage. After a_p and a_q are removed, then nothing prevents the removal of actions a_i and a_j.

The above ideas can be applied to longer sequences of actions but the complexity will rise. If we deal with k-tuples, then the number of these might be up to n^k. Moreover, in this case we have to look if we can replace these k-tuples of actions with at most $k-1$-tuples of (different) actions which can be done in $O(|A|^{k-1})$ steps. Therefore, it seems to be unrealistic to use an exhaustive approach (i.e., testing all the possible k-tuples of actions). However, it might be appropriate to have some sort of knowledge base pointing out where to find non-optimalities and how to deal with them. How to obtain such a knowledge base is a subject of ongoing work.

Case Studies

We have observed some positive aspects of our approach. Many inverse actions are revealed and removed (including 'nested' inverse actions) despite being non-adjacent in plans. For instance, in 'BlockWorld'-like domains (e.g., Depots) we are able to detect situations where we at first build a tower of blocks and after that we tear down a part of it, which is unnecessary. The detection is still possible even if we have some interleaving actions in the plan that do not affect the situation around the tower of blocks. Otherwise, the detection might not be possible (explained later). Another good example, where our method works, is consecutive application of the DRIVE operator on the same truck (Depots domain). If $drive(t1, l1, l2)$ and $drive(t1, l2, l3)$ are weakly adjacent (i.e., no crate is loaded or unloaded from $t1$ in $l2$), then we can replace them by $drive(t1, l1, l3)$.

There is another positive aspect of using our method. If we enhance domains by macro-operators (operators defined as a normal planning operator but behaving as a sequence of primitive operators), then even an optimal planner may fail to find an optimal solution. This is because a 'classical' planner deals with macro-operators in the same way as primitive operators even though the macro-operators are more 'expensive' (consist of more primitive operators). One possibility is to add an action cost where the cost of macro-operators is a sum of costs of the primitive operators. However, this approach is more difficult for planners. Our approach is able to detect some non-optimalities caused by using macro-operators, for instance, if a planner uses a sequence of macro-operators $\langle$LIFT-LOAD,UNLOAD-DROP$\rangle$ rather than $\langle$LIFT,DROP$\rangle$ (these need not be inverses, e.g. if we drop a crate onto a different pallet than that from which it was lifted), then we can see that both sequences have the same length but if we unfold the first one (e.g., replace macro-actions by corresponding primitive ones), then the the first sequence is longer. After the first sequence is unfolded we can see that LOAD and UNLOAD are inverse and can be removed (by our method) and we obtain $\langle$LIFT,DROP$\rangle$ which is optimal. In a hypothetical case

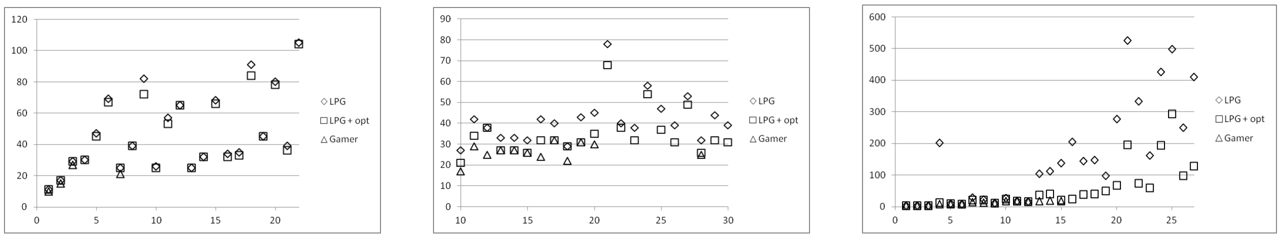

Figure 1: Experimental results. The Depots domain on the left hand side, the Gold-Miner domain in the middle and the Storage domain on the right hand side. x-axis refers to problems and y-axis refers to plan lengths.

where a planner uses a sequence $\langle a, a_{i,j} \rangle$ ($a_{i,j}$ is a macro-action assembled from a_i and a_j) but a sequence $\langle a, a_i \rangle$ can be replaced by an action a', then our approach can detect it and provide a sequence $\langle a', a_j \rangle$. However, optimal planning with macro-operators is a subject of ongoing work.

Some examples are sufficient to show that our approach does not guarantee optimality. E.g., if in the Depots domain we build a tower of blocks, then use the same hoist for moving another block and after that we tear down a part of the tower, then we are unable to detect inverse actions. This is because there is a dependence between the action stacking the last block on the tower and the action picking up the other block (because the stack action frees the hoist which is a precondition of the pick up action). This also holds for the action stacking the other block and the action unstacking the top block from the tower. The main reason why our approach does not work in this case is that we consider dependencies (and independencies) between single actions and not action sequences. Considering dependencies between action sequences (in fact macro-actions) should handle this issue, however, it may be harder (in terms of complexity). Our approach also cannot deal with situations where some subplan longer than two is replaceable by another shorter subplan. For instance, if we unload a crate from one truck and load the crate on the second truck. It is clearly better to make an extra drive with the first truck (one action) rather than move the crate to the second truck (two actions).

Preliminary Experimental Evaluation

For our experiments we chose five benchmark domains (typed STRIPS) and corresponding problems, namely Depots (all), Gold-Miner (target-typed 10 – 30), Satellite (all), Zeno (all) and Storage (1-27), from the International Planning Competition (IPC)[1] and planners, namely LPG-td (Gerevini, Saetti, and Serina 2004) and GAMER (Edelkamp and Kissmann 2008), that successfully participated in the IPC. LPG-td is based on using greedy local search techniques on Planning Graph which often finds solutions in a short time but they are of low quality. On the other hand GAMER is based on exploring Binary Decision Diagrams and guarantees optimal solutions, however, the running time is usually very high. Both the planners were run on Intel i5 2.8 GHz, 8GB RAM, Ubuntu Linux. Our method for plan optimization is written in C++ and was run on the same machine but under Windows 7.

The results (see Figure 1) depict how our method affects lengths of plans found by LPG in comparison with optimal plans (the shortest plans) found by GAMER. GAMER was able to solve only 4 problems in the Depots domain, 12 problems in the Gold-Miner domain and 15 problems in the Storage domain[2]. In the Depots domain LPG produced plans in a good quality though there were almost no space for optimization. On the other hand, in two cases the optimization was quite significant. In the Gold-Miner our method achieved very promising results (plans were shortened by more than 16%) because in almost all cases it was able to reduce plan lengths quite significantly and in several cases the optimization process led towards optimal solutions. The best results were achieved in the Storage domain, where the plans were shortened by approximately 63% ! Especially, solutions of harder problems (13-27) were optimized significantly. In the Zeno and Satellite domains (not graphically depicted) our method shortened the plans found by LPG by 4-5% (similar to the Depots domain).

The running time of LPG-td (the speed option) was of the order of tens of millisecond per problem. Even though our implementation is not optimized for performance the running time of our method stays within the second per domain range, only in the Storage domain the time exceeds three seconds. It gives us approximately 20-50ms per problem.

Conclusions

We presented a method for plan optimization which is based on investigating action dependencies and independencies. The method is able to optimize plans in a little time thus it is useful in combination with a fast but not very optimal planner (e.g. LPG). Though a tractable method for obtaining optimality in every case is impossible, our method performed well in the preliminary empirical evaluation, and executed in a time comparable to the time of initial plan generation.

In our future work we aim to investigate how the method (or an extension of it) can shorten plans produced by planners using macro-operators. Moreover, we aim to investigate techniques for forming knowledge bases that will help us to detect common non-optimalities in plans (discussed above) and how to fix them.

[1]http://ipc.icaps-conference.org

[2]time limit set as 15 minutes

References

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90(1-2):281–300.

Bylander, T. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence* 69:165–204.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Chrpa, L., and Barták, R. 2008. Towards getting domain knowledge: Plans analysis through investigation of actions dependencies. In *Proceedings of FLAIRS 2008*, 531–536.

Chrpa, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review* 25(3):281–297.

Edelkamp, S., and Kissmann, P. 2008. Gamer: Bridging planning and general game playing with symbolic search. In *Proceedings of the sixth IPC*.

Fink, E., and Yang, Q. 1992. Formalizing plan justifications. In *In Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, 9–14.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 956–961.

Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning in pddl2.2 domains with lpg-td. In *Proceedings of the fourth IPC*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning, theory and practice*. Morgan Kaufmann Publishers.

Helmert, M. 2003. Complexity results for standard benchmark domains in planning. *Artificial Intelligence* 143(2):219–262.

Helmert, M. 2006. New complexity results for classical planning benchmarks. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006, Cumbria, UK*, 52–62.

Nakhost, H., and Müller, M. 2010. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In *Proceedings of ICAPS*, 121–128.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Westerberg, C. H., and Levine, J. 2001. Optimising plans using genetic programming. In *Proceedings of ECP*, 423–428.