Reformulating Planning Problems by Eliminating Unpromising Actions

Lukáš Chrpa and Roman Barták
Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague
{chrpa,bartak}@ktiml.mff.cuni.cz

Abstract

Despite a big progress in solving planning problems, more complex problems still remain hard and challenging for existing planners. One of the most promising research directions is exploiting knowledge engineering techniques such as (re)formulating the planning problem to be easier to solve for existing planners. In particular, it is possible to automatically gather knowledge from toy planning problems and exploit this knowledge when solving more complex planning problems. In this paper we propose a method for eliminating some actions from the problem specification that are often useless or may mislead the planners. The method detects if actions are somehow connected with the initial or goal predicates and by using this information we suggest that some actions are not necessary when solving the planning problem. To eliminate these actions we modify the planning domain and hence the method remains independent of used planning system.

Introduction

AI (Artificial Intelligence) planning is a traditional and still hot research topic thanks to its theoretical interest as well as wide real-life applicability ranging from robotics to web service composition. Despite the significant improvement of planning system in recent years many planning problems still remain hard and challenging. This opens up a lot of possibilities for research.

In the last decade, a lot of planning systems were developed and many of them competed in the International Planning Competition (IPC)[1]. Along with those planning systems, many novel planning techniques have been proposed. Advanced planning techniques are usually based on the Planning Graph (Blum & Furst 1997) accompanied by many different and mostly powerful heuristics (Bonet & Geffner 1999). In addition, some planners benefit from a translation of a planning problem into other formalisms such as Boolean Satisfiability (SAT) (Kautz & Selman 1992) or Constraint Satisfaction Problems (CSP) (Do & Kambhampati 2001).

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

However, even the best planners often fail to find a solution in a reasonable time if the planning problem is more complex. Theoretically, it has been proven by (Erol, Nau, & Subrahmanian 1995) that the complexity of plan existence is EXPSPACE-complete in the classical representation of planning problems (Ghallab, Nau, & Traverso 2004). Even though powerful heuristics are used to guide planners, it is almost impossible to confront such a complexity with 'brute force' search methods only. It seems to be more than reasonable to gather knowledge that should help planners to significantly reduce the depth of the search space as well as the branching factor. Such knowledge can be learnt from given training plans that are obtained as solutions of simpler planning problems.

One of the well known sorts of knowledge are the so called macro-operators (Korf 1985) – operators representing a sequence of other (primitive) operators and behaving like a single operator. There were developed several approaches (Coles, Fox, & Smith 2007; Newton, Levine, & Fox 2005) reaching promising results. One of the most influential works in this area is a planning system called Macro-FF (Botea *et al.* 2005) that combines its own macro-operator learner and the well known planner FF (Hoffmann & Nebel 2001). Even though macro-operators are good for reducing the depth of the search space, they are usually responsible for a significant increase of the branching factor. The high branching factor is a challenge we are facing in this paper. Clearly, if there are many instances of operators (actions) in the planning process then the planning process tends to be slow and inefficient, because in every node it may be forced to choose among many alternatives. It is not a surprise that the number of operators' instances (actions) may grow exponentially depending on the number of operators' arguments and the number of objects listed in a particular planning problem.

Existing planning systems usually do not generate all instances of planning operators in preprocessing. For example, the FF planner (Hoffmann & Nebel 2001) instantiates only reachable actions - actions that can be applicable at some point of the planning process. The work presented in (Helmert 2006) uses the idea called reach-one-goal (i.e. achieve the goals of the planning task consecutively), where the solver focuses on such actions that may be relevant for a particular goal.

In this paper we investigate the connectivity between some planning operators and initial or goal predicates. We propose a method that is able to detect such a connectivity from given training plans and encode it back into planning domains and their related planning problems. Hence the transformed domains and problems can be easily passed to existing planners. The main contribution of this paper rests in pruning actions that may mislead the planners when searching for a solution. In comparison to works mentioned above that prune only unreachable actions, our approach can prune also actions that are normally reachable but (mostly) unnecessary.

The paper is organized as follows. First, we will introduce basic notions from the planning theory followed by a motivating example. Then we will give the theoretical background of the proposed approach. After that, we will describe the method for detecting connectivity between planning operators and initial or goal predicates. We will also suggest how such a connectivity can be encoded into planning domains and problems. Finally, we will present the experimental evaluation of the method and in conclusions we will discuss possible future work.

Preliminaries

Traditionally, AI planning deals with the problem of finding a sequence of actions transforming the world from some initial state to a goal state. **Predicate** $p(t_1, \dots, t_n)$ is a construct, where p represents a predicate symbol and $t_1, \dots, t_n$ are terms (variables or constants). A predicate is **grounded** if and only if all its terms are constants. **State** s is a set of (grounded) predicates that are true in s. **Action** a is a 3-tuple $(pre(a), eff^-(a), eff^+(a))$ of sets of grounded predicates such that $pre(a)$ is a set of grounded predicates representing the precondition of action a, $eff^-(a)$ is a set of negative effects of action a, $eff^+(a)$ is a set of positive effects of action a, and $eff^-(a) \cap eff^+(a) = \emptyset$. Action a is applicable to state s if $pre(a) \subseteq s$. If action a is applicable to state s, then the new state s' obtained after applying the action is $s' = (s \setminus eff^-(a)) \cup eff^+(a)$. **Planning operator** o is a 3-tuple $(pre(o), eff^-(o), eff^+(o))$ of sets of predicates (not necessarily grounded) and if there exists a substitution Θ from variables to constants then $(pre(o), eff^-(o), eff^+(o))\Theta$ is an action. A **planning domain** is represented by a set of predicates and a set of operators (or actions if the planning domain is grounded). A **planning problem** is represented by a planning domain, a set of constants representing particular objects, an initial state, and a set of goal predicates. A (valid) **plan** is an ordered sequence of actions which leads from the initial state to any goal state containing all of the goal predicates. A planning problem is **solvable** if and only if there exists a valid plan. For deeper insight in this area, see (Ghallab, Nau, & Traverso 2004).

Motivating example

Let us consider the well known planning domain Blocksworld (Slaney & Thiébaux 2001), where the robotic arm builds towers from available blocks. We focus on the version with four operators: STACK (stacks a block on another block), UNSTACK (unstacks a block from another block), PICKUP (picks a block from the table), and PUTDOWN (puts a block to the table). Operators STACK and UNSTACK have two arguments (they operate with two blocks), the other operators just one. It is easy to find out that for problems considering a hundred blocks, the operators STACK and UNSTACK have almost ten thousand instances and all of them are reachable. However, if we observe the Blocksworld domain more deeply then we can reveal the following facts. At first we have to take down the initial 'towers' of blocks (or necessary parts of them) and put the blocks on the table. At second we have to build the required 'towers' of blocks. Now, we can see that we need only such instances of UNSTACK (resp. STACK) operators having a connectivity with the initial (resp. goal) predicates describing what block is on another block. In particular, operator UNSTACK(X,Y) requires block X to be stacked on block Y. If some block A is initially stacked on block B then we allow action UNSTACK(A,B), otherwise we deny this action. This idea can be analogically applied to operator STACK. By this approach we are able to prune many instances of STACK and UNSTACK operators that are not necessary when solving any planning problem in this domain.

Theoretical background

Planning systems process planning domains with corresponding planning problems to produce their solutions, plans. Informally said, planning domains serve like abstract templates for given environments, and planning problems, on the other hand, describe those environments concretely and define certain planning tasks. The main challenge for planning is the exploration of a huge search space, whose size depends directly on the number of applicable actions. We focus on the problem of excessive number of actions that may mislead planners when looking for solutions. If we consider an operator with arity s and a planning problem with n (untyped) objects then the number of all possible instances of this operator is n^s. Often, this is simply too many actions to be considered during the planning process. In fact, many of these actions may be useless and not all these actions can be pruned by checking their reachability using the methods mentioned in the introduction. Our goal is to reduce the number of such useless actions (especially reachable) by analyzing simpler plans.

We found out that in some cases there exists a connection between operators' predicates in preconditions and initial predicates or operators' predicates in positive effects and goal predicates. Now, we formally define several notions describing these kinds of connections.

Definition 1.1: Let $P = \langle \Sigma, s_0, g \rangle$ be a planning problem, o be a planning operator from Σ and p be a predicate. Operator o is *entangled by init (resp. goal)* with predicate p in planning problem P if and only if $p \in pre(o)$ (resp. $p \in eff^+(o)$) and there exists a plan π that solves P and for every action $a \in \pi$ which is an instance of o and for every grounded instance p_{ground} of the predicate p holds: $p_{ground} \in pre(a) \Rightarrow p_{ground} \in s_0$ (resp. $p_{ground} \in eff^+(a) \Rightarrow p_{ground} \in g$).

Definition 1.2: Let Σ be a planning domain, o be a planning operator from Σ and p be a predicate. Operator o is *fully entangled by init (resp. goal)* with predicate p if and only if there does not exist any planning problem P over Σ where o in not entangled by init (resp. goal) with p in P. In addition we define a set $plans(P, o, p, init$ (resp. $goal)) = \{\pi \mid \pi$ is a solution of P and satisfy the conditions of entanglement of o and p regarding def. 1.1$\}$.

The entanglement by init (resp. goal) says that in a particular planning problem we use only such instances of operators, whose predicates in preconditions (resp. positive effects) correspond with the initial (resp. goal) predicates. The full entanglement extends this for every solvable planning problem in a particular domain.

Recall our motivating example from the previous section. Assume that binary predicate $on(X, Y)$ represents the fact that block X is stacked on block Y. Now, we can easily see that the operator UNSTACK is fully entangled by init with predicate on. It means that we unstack blocks only from their initial positions. Analogically, the operator STACK is fully entangled by goal with predicate on. It means that we stack blocks only to their goal positions.

In the following paragraphs we shall show that all static predicates (predicates that do not appear in effects of any operator) with respect to operators having these predicates in preconditions satisfy the conditions to be fully entangled by init.

Definition 1.3: Let Σ be a planning domain and p be a predicate such that there does not exist any substitution Θ and any operator o belonging to Σ such that $p\Theta \in eff^-(o)$ or $p\Theta \in eff^+(o)$ (in the other words $p\Theta$ represents a variant of p). Then p is a *static predicate* with respect to Σ.

Proposition 1.4: Let Σ be a planning domain and p be a static predicate (with respect to Σ). Then for every operator o belonging to Σ where there exists a substitution Ψ such that $p\Psi \in pre(o)$, it holds that o is fully entangled by init with p and for every planning problem P $plans(P, o, p, init)$ contains all plans that solve P.

Proof: Let $P = \langle \Sigma, s_0, g \rangle$ be an arbitrary solvable planning problem and p be a static predicate with respect to Σ. Let s be an arbitrary state reachable from s_0 which means that there exists a valid sequence of actions (instances of operators from Σ) transforming state s_0 to s. We shall show that all instances of p from s_0 belong also to s and no other instance of p can belong to s. From the assumption we know that no variant of p appears in positive or negative effects of any operator from Σ. It means that we cannot add or remove any instance of p from s_0 which implies that in any reachable state from s_0 we have the same instances of p. Consequently for any action a in the above valid sequence of actions starting in s_0, it holds $p_{ground} \in pre(a) \Rightarrow p_{ground} \in s_0$ (otherwise the action is not applicable to a reachable state). Hence, operator o from Σ giving these actions as its instances is entangled by init with predicate p in problem P. As P can be an arbitrary planning problem in domain Σ, operator o is fully entangled by init with p. It is also clear that $plans(P, o, p, init)$ contains all plans that solves P. $\Box$

Definition 1.2 ensures the existence of plans for every solvable planning problem when pruning actions violating the full entanglement conditions. For static predicates as we proved before, the sets of plans solving particular (solvable) planning problems remain the same. In general case, each full entanglement somehow restricts those sets of plans allowing only such actions that do not violate the particular full entanglement. However, such restrictions differ regarding the particular full entanglements. The different full entanglements can be used together in such a way that every solvable planning problem remains solvable even if actions violating at least one of the full entanglements are pruned.

Definition 1.5: Let Σ be a planning domain, *SFE* a set of triples $SFE = \{(o_1, p_1, t_1), \ldots, (o_n, p_n, t_n)\}$, where o_i is an operator from Σ, p_i is a predicate from Σ and $t_i \in \{init, goal\}$ (i.e., o_i is fully entangled by t_i on p_i). If for every solvable planning problem P over Σ $\bigcap_{i=1}^{n} plans(P, o_i, p_i, t_i) \neq \emptyset$ holds, then SFE is *a set of compatible full entanglements*.

The following proposition formally describes how the full entanglement affects possible operators instances (actions).

Proposition 1.6: Let Σ be a planning domain, $P = \langle \Sigma, s_0, g \rangle$ be an arbitrary planning problem that is solvable and $SFE = \{(o_1, p_1, t_1), \ldots, (o_n, p_n, t_n)\}$ be a set of compatible full entanglements. Let A be a set of all actions a meeting the following conditions:

1. a is an instance of planning operator o from Σ
2. if $(o, p, init) \in SFE$ then $\forall \Theta : p\Theta \in pre(a) \Rightarrow p\Theta \in s_0$
3. if $(o, p, goal) \in SFE$ then $\forall \Theta : p\Theta \in eff^+(a) \Rightarrow p\Theta \in g$

Let Σ' be a grounded planning domain containing set of actions A and a corresponding set of grounded predicates. Then, planning problem $P' = \langle \Sigma', s_0, g \rangle$ is solvable and a plan for P' is a plan for P too.

Proof: We have to prove that we the reformulated planning problem P' is solvable and a plan for P' is also a plan for P. We can simply see that every plan for P' contains only actions from A. As we can see from the assumption, actions from A meet three conditions. For the first condition, it is clear that any solution of P contains only such actions that are instances of the operators defined in Σ. For the second condition, we know that if any operator o is fully entangled by init with any predicate p then we allow only such instances of o having in their preconditions such instances of p that correspond with instances of p belonging to the initial state. From definition 1.1 (entanglement) we know that there exists a plan π containing such actions satisfying the following conditions. If action $a \in \pi$ is an instance of operator o which is entangled by init with any predicate p in P then $\forall \Theta : p\Theta \in pre(a) \Rightarrow p\Theta \in s_0$ (Θ is a substitution from variables to constants, so $p\Theta$ is a grounded predicate). Analogically it holds for the third condition. Considering SFE is a set of compatible full entanglements, we know that there exists at least one plan which satisfy the entanglement conditions (def. 1.1) for all operators and predicates that are fully entangled. Now it is clear that P' is also solvable. □

Informally said, if some operator is fully entangled on some predicate (or predicates) then we can omit such instances of this operator where the predicate (or predicates) in the preconditions or positive effects are not corresponding with the particular predicates belonging to the initial or goal state. We assume that by detecting (compatible) full entanglements we can omit a lot of unnecessary actions. Full entanglements can be understood as a piece of knowledge that we can pass to existing planners via the definition of the planning problem without updating the source code of the planner. There are (at least) two ways how the information about full entanglements can be encoded in the planning domain/problem. The first option rests in production of grounded domains (and problems) as we did it in Proposition 1.6. The main disadvantage of this approach rests in the fact that the grounded domains may be very large which may cause a significant slow-down of the planner in the pre-processing phase (loading the problem). It may also disallow some other techniques such as lifting. The second option rests in the extension of planning domains by special static predicates. We shall present this approach in the following paragraphs.

Definition 1.7: Let Σ be a planning domain and $P = \langle \Sigma, s_0, g \rangle$ be an arbitrary planning problem. If operator o from Σ is fully entangled by init (resp. goal) with predicate p and p is not a static predicate then we define *reformulated domain* $Ref(\Sigma, o, p)$ and *reformulated problem* $Ref(P, o, p)$ in the following way:

1. create a new predicate p' which is not present in Σ and has the same arity as p
2. create an operator $o' = (pre'(o), eff^-(o), eff^+(o))$, where $pre'(o) = pre(o) \cup \{p'\}$ (the arguments of p' correspond with the arguments of p)
3. create a reformulated domain $Ref(\Sigma, o, p)$ from Σ by adding p' and replacing o by o'
4. create a reformulated problem $Ref(P, o, p) = \langle Ref(\Sigma, o, p), s'_0, g \rangle$, where $s'_0 = s_0 \cup \{p'\Theta \mid p\Theta \in s_0 \text{ (resp. } p\Theta \in g)\}$

Theorem 1.8: Let $P = \langle \Sigma, s_0, g \rangle$ be a planning problem and $SFE = \{(o_1, p_1, t_1), \ldots, (o_n, p_n, t_n)\}$ be a set of compatible full entanglements. Let P' be a problem obtained from P by successive application of $Ref(P, o, p)$ for every $(o, p, t) \in SFE$, where p is a non-static predicate. Then, if P is solvable then P' is solvable and a plan for P' is a plan for P too.

Proof: Recall Definition 1.7, where $Ref(P, o, p)$ was introduced. We added the predicate p' into the reformulated domain and problem. It is clear that p' is a static predicate, because it does not appear in effects of any operator. From the proof of Proposition 1.4 we know that static predicates hold through the whole planning process without being changed. It can be easily observed that no action with instance of p' in the precondition that do not correspond with the initial instances of p' can be applied to any state. From the 2nd and 4th point (Def. 1.7) we can see that arguments of p' correspond with the arguments of p. From Proposition 1.6 we know that if we remove such operators' instances from the domain that 'break' full entanglements from SFE (recall conditions 2 and 3 from Proposition 1.6), then it does not affect the solvability of the reformulated problem.□

Detection of full entanglements

In the previous section, we showed that if we know full entanglement relations then we can restrict the set of grounded actions (Definition 1.7 and Theorem 1.8) to be assumed during planning without affecting solvability of the planning problem. The remaining question is how the (compatible) full entanglements can be detected. Theoretically, we are facing the problem of validating entanglements for all solvable planning problems over a given domain. For the static predicates the detection is easy and can be realized for every domain (Proposition 1.4). For the other predicates we have two options. First, we can prove (compatible) full entanglements theoretically (it is not practically possible to explore all solvable planning problems) or second, we can use heuristics to guess that a given operator can be fully entangled with a given predicate by exploring only a fraction of planning problems (training planning problems). In the first case we can fully exploit the theoretical results from the previous section and we are sure that we cannot break the solvability of planning problems by pruning of actions mentioned above. In the second case we cannot assure that the solvability is preserved because the heuristic method does not guarantee full entanglement.

In our opinion, the first option may require domain-dependent approaches and hence a domain expert who can handle it. Therefore, in the rest of the paper we will focus on the second option with the goal to have a fully automated domain-independent approach. It motivates us to develop a method for detection of full entanglement based on heuristics.

The main idea of our approach is as follows. Instead of exploring all planning problems and finding a plan validating the condition from Definition 1.1, we assume only a subset of training problems together with existing plans for these problems. The entanglement condition as specified in Definition 1.1 is checked only for these training problems and plans and if validated we declare full entanglement as the following heuristic specifies.

```
INPUT: planning domain $\Sigma$, set of planning problems and their plans
OUTPUT: a set of compatible full entanglements
 Set the full entanglements by init (resp. goal) between all operators and predicates
   from the corresponding preconditions (resp. positive effects) from $\Sigma$
 ForEach planning problem $P$ do
   ForEach operator $o$ from $\Sigma$ do
     ForEach $p \in pre(o)$ do
       ForEach $a \in \pi$ where $a$ is an instance of $o$ and $\pi$ is a plan solving $P$ do
         If $\neg\exists\Theta : p\Theta \in s_0 \wedge p\Theta \in pre(a)$ where $s_0$ is an initial state of $P$
         then Unset the full entanglement by init between $o$ and $p$
       EndForeach
     EndForeach
     ForEach $p \in eff^+(o)$ do
       ForEach $a \in \pi$ where $a$ is an instance of $o$ and $\pi$ is a plan solving $P$ do
         If $\neg\exists\Theta : p\Theta \in g \wedge p\Theta \in eff^+(a)$ where $g$ is a set of goal predicates
         of $P$ then Unset the full entanglement by goal between $o$ and $p$
       EndForeach
     EndForeach
   EndForeach
 EndForeach
```

Figure 1: Algorithm for heuristic detection of the full entanglements by init or by goal.

Heuristic 2.1: Let Σ be a planning domain. If for each training planning problem P over Σ together with a plan solving the problem it holds that operator o from Σ is entangled by init (resp. goal) on predicate p then operator o is considered as fully entangled by init (resp. goal) with predicate p. We also consider that all detected full entanglements are compatible.

This heuristic gives us an opportunity to develop the algorithm (fig. 1) for detection of full entanglement (by init or goal) at the cost of losing its completeness. It means that the algorithm may declare full entanglement even if it does not hold.

The algorithm starts with an assumption that every operator from given domain is fully entangled by init (resp. goal) with every predicate listed in the corresponding preconditions (resp. positive effects). The algorithm is testing if the conditions of entanglement are satisfied in each training plan for every operator and the corresponding predicate. If the condition of entanglement is broken (once is enough) then we set the particular pair (operator and predicate) as not fully entangled.

The worst-case time complexity of the algorithm can be estimated in the following way. Let n be the number of training planning problems $P_i = \langle \Sigma, s_0^i, g^i \rangle$ and π_i be a plan solving P_i. Let O be the set of operators from Σ and $\wp$ be the set of predicates from Σ. Then the time complexity of the algorithm is following:

$$O\left(|O||\wp|\sum_{i=1}^{n}(|\pi_i|)\left(\sum_{i=1}^{n}(|s_0^i|)+\sum_{i=1}^{n}(|g^i|)\right)\right)$$

In most current planning domains and their corresponding planning problems it holds that $|O| \ll \sum_{i=1}^{n}(|\pi_i|)$, $|\wp| \ll \sum_{i=1}^{n}(|\pi_i|)$ and $|g^i| \ll |s_0^i|$. Hence we can update the time complexity estimation in the following way:

$$O\left(\sum_{i=1}^{n}(|\pi_i|)\sum_{i=1}^{n}(|s_0^i|)\right)$$

The low time complexity of the algorithm means that we can run the algorithm even for more training problems. However, we need to consider that every training problem must be solved before the algorithm can start computation (we need a training plan). Even though there are good planners around, solving of many training problems can still be very time consuming. It means that we should use ‘reasonable‘ training plans - usually toy problems - only.

Moreover, the existing planners frequently do not produce shortest plans even though some of them successfully participated in optimal tracks of IPC! This may cause problems to our heuristic algorithm because if the training plan contains actions that are not necessary to solve the problem, these actions may break the condition of full entanglement. Hence we suggest to weaken the heuristic further to allow ‘a few‘ violations of the entanglement condition.

Heuristic 2.2: Let Σ be a planning domain. If for each training planning problem P over Σ holds that operator o from Σ is NOT entangled by init (resp. goal) on predicate p in less or equal than $n\%$ times (‘flaws‘ ratio), then operator o is considered as fully entangled by init (resp. goal) with predicate p. We also consider that all detected full entanglements are compatible.

The ‘flaws‘ ratio describes the ratio between the number of violations of full entanglements and the total number of instances of a particular operator in all training plans. To follow heuristic 2.2 the detection algorithm can be updated in such a way that instead of unsetting of full entanglements we increase the number of violations of these full entanglements and decide about the full entanglement at the end based on the ‘flaws‘ ratio. Clearly, the risk of ‘false positive‘ detection of full entanglements is getting higher as the ‘flaws‘ ratio raises.

‘Flaws‘ ratio introduces a parameter to the algorithm which raises the question how to set this parameter. In the lines bellow we shall present an approach which can help to determine the ‘flaws‘ ratio.

1. set $flaws$ to n, where $n \in (0; 1)$; according to our experiments we suggest starting with n = 0.1
2. generate the entanglements by the modified algorithm using ‘flaws‘ ratio $flaws$
3. compare the generated entanglements to the entanglements obtained by the original algorithm (without ‘flaws‘). If same then quit (we are not able to gather additional entanglements).
4. generate a reformulated domain and reformulated training problems considering the generated entanglements (according to Definition 1.6)
5. run the planner on all the reformulated training problems. If succeed then quit (we found proper full entanglements with respect to training plans).
6. otherwise decrease $flaws$ by ϵ ($\epsilon > 0$, for example ϵ = 0.01) and go to the second step.

Problem	no fr		with fr	
	Unary	Binary	Unary	Binary
Depots	1(2)	2(2)	1(2)	3(3)
Driverlog	1(1)	2(2)	1(1)	2(3)
Storage	2(2)	1(1)	2(2)	1(1)
Zenotravel	0(0)	2(2)	0(0)	2(2)
GoldMiner	3(3)	0(0)	3(3)	0(0)
MatchingBW	5(9)	2(4)	5(9)	3(5)
Parking	3(4)	2(2)	3(4)	2(2)
Thoughtful	15(47)	3(6)	15(47)	3(6)

Table 1: The table shows how many unary and binary predicates were added and how how many times they were added to operators' preconditions (in brackets).

Experimental evaluation

We evaluated our approach experimentally in the following way. At first, we looked for what our methods can learn (how many new predicates are added to modified domains by applying Definition 1.6). At second, we compared running times and plan lengths of well known planners SATPLAN 2006 (Kautz, Selman, & Hoffmann 2006) (winner of the 5th IPC optimal track), SGPLAN 5.22 (Hsu *et al.* 2007) (winner of the 5th IPC sub-optimal track) and LAMA (Richter & Westphal 2008) (winner on the 6th IPC sub-optimal track) on a couple of planning domains well known from the IPC. In summary, we proceeded the evaluation in the following steps:

- generate several simpler training plans (by SATPLAN),
- run our methods (with and without 'flaws' ratio) for detection of entanglements,
- generate reformulated domains and problems considering the detected entanglements,
- run the planners both on the original problems and on the reformulated problems and compare results.

We used several planning domains for our evaluation to ensure that the proposed approach is generally applicable (rather than specific for a particular planning domain). In particular, we used domains $Depots$, $DriverLog$, and $ZenoTravel$ from the third IPC, domain $Storage$ from the fifth IPC, and domains $GoldMiner$, $MatchingBlockWorld$, $Parking$, and $Thoughtful$ from the learning track of the sixth IPC.

Learning phase

For the learning phase we used 3 to 6 training planning problems depending on the particular domain. For generation of training plans we used SATPLAN. The selected training planning problems were not too complex which results that the training plans were generated mostly within tenths of seconds. The detection of the full entanglement with generation of reformulated domain and problems took at most half of second[2].

In table 1 it is shown how many unary and binary predicates were added and how how many times they were added to operators' preconditions (in brackets). We focused only on unary and binary predicates, because nullary predicates did not bring any useful information and ternary predicates (or more) were not presented in the tested domains (or never detected as fully entangled). We also compared both methods - without and with 'flaws' ratio. We found out that using the 'flaws' ratio contributed in Depots, Driver Log and Matching BlockWorld domains. The other domains remained the same as reformulated by the method without 'flaws' ratio.

Running times and plans lengths comparison

The results of the evaluation are showed in table 2 (plan lengths are in brackets)[3]. Symbol '-' in the cells of the table was used only in columns representing reformulated problems by the method with 'flaws' ratio where it indicates the fact that this method did not reveal more knowledge than the method without 'flaws' ratio (see the previous subsection). Term 'err' in the table cells means that the planner terminated with unexpected error (probably caused by some planner's bug). All reformulated problems remained solvable except thoughtful-s7-t5b and thoughtful-s7-t5c. This was caused by the fact that the (toy) training problems for Thoughtful domain took in account full entanglements that were applicable for the problems of type thoughtful-s5-t4, but too restrictive for the problems of type thoughtful-s7-t5.

The best results were acquired by SATPLAN. Running times of reformulated problems (especially those that were generated by the method with 'flaws' ratio) were (mostly significantly) better in all tested cases. The plan quality (the less plan length the better) were also in most cases better in the reformulated problems. Even though SATPLAN produces optimal plans in makespan, it does not ensure the optimality with respect to the number of actions. SATPLAN simply find the first plan with the lowest makespan. The reformulated problems operate with the lesser number of actions than the original ones. It may result in the better plan quality for the reformulated problems.

SGPLAN's running times of the reformulated problems were mostly better than the original ones (especially in the reformulated Matching BlockWorld domain and Parking domain). However, in some cases the running times of the reformulated problems were significantly worse. The reason of this may lie in SGPLAN's heuristics that are based on heuristics used in Metric-FF. FF based heuristics are vulnerable to problems that may contain dead-ends (i.e. we can reach such a state from which the goal is unreachable). Pruning of actions done by our methods may cause that some problems become dead-ended. The quality of plans was better in most of the reformulated Depots problems, slightly worse in the reformulated Storage problems and significantly worse in the reformulated Parking domain.

LAMA's running times of the reformulated problems were mostly better than the original ones. However, similarly to SGPLAN, the running times of several reformulated problems were worse. The reason of this also may rest in LAMA's heuristics (based on FF and Causal Graph) that

[2]performed on Core2Duo 2.66GHz, 4GB RAM, Win XP

[3]performed on XEON 2.4GHz, 1GB RAM, Ubuntu Linux

may be vulnerable to problems with dead-ends. The most interesting and a bit surprising result was achieved in Gold Miner domain, where the quality of solutions of reformulated problems were much better than in original ones.

Additional remarks

The presented results showed that our approach is reasonable and can help planners increase their performance. SATPLAN, which gained the best results, is based on transforming of the planning graph (Blum & Furst 1997) into a SAT formulae. We suppose that SATPLAN benefits from our methods because our methods result in a significant reduction of the size of the planning graph. Planners like SGPLAN or LAMA using FF-based heuristics may occasionally experience difficulties when using our approach. It was discussed in the previous subsection that the main problem of this (we suppose) rests in the fact that problems reformulated by our methods may contain dead-ends. On the other hand in most of problems our method is still helpful. For instance, Matching BlockWorld domain is a good example of problems with dead-ends. As we anticipated, SGPLAN experienced difficulties when solving original problems. Our methods helped SGPLAN to solve these problems, often in a very good time. It shows that our methods can be successfully used in a connection with planners based on FF-based heuristics (like SGPLAN) mostly for problems with dead-ends. In addition, many planning problems, especially real world ones, have dead-ends.

Conclusions

In this paper, we presented methods for domain and problem transformations that can prune many unnecessary actions that may mislead planners when solving the planning problem. The proposed methods are based on detection of connectivity (here defined as full entanglements) between operators' instances in training plans and initial or goal predicates in corresponding planning problems. The main advantage of the proposed approach rests in a possible reduction of searching space. The presented experimental evaluation confirmed that in most cases our approach reduced the time needed to find a solution.

Another advantage lies in independence of the proposed techniques from the planning algorithm. In particular, we suggest modification of planning domains and problems, so that it is not required to modify the planners themselves. Hence our approach can be combined with other pre-processing techniques such as reachability analysis.

A possible mutual reinforcement rests in connection of our approach with methods for learning macro-operators (Botea *et al.* 2005). As we mentioned before, using of macro-operators reduces the depth of search trees at the cost of increase of the branching factor. We believe that in connection with our method we can reduce back the branching factor which may result in significant improvement of the whole planning process.

We also plan to study other possibilities for reducing the number of generated instances of operators. We know that the full entanglements by goal we are usually limited by the lesser number of goal predicates. We would like to study possibilities how to extend our approach by detecting of such predicates that must be also true in any goal state (though not specified as goal predicates). It may help us to find more relations of full entanglement by goal.

Acknowledgements

The research is supported by the Czech Science Foundation under the contract no. 201/08/0509.

References

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

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *Proceedings of ECP*, 360–372.

Botea, A.; Enzenberger, M.; Muller, M.; and Schaeffer, J. 2005. Macro-ff: Improving ai planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24:581–621.

Coles, A.; Fox, M.; and Smith, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, 97–104.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into csp. *Artificial Intelligence* 132:151–182.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* 76:75–88.

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

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.

Hsu, C.-W.; Wah, B. W.; Huang, R.; and Chen, Y. 2007. *SGPlan*. http://manip.crhc.uiuc.edu/programs/SGPlan/index.html.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *Proceedings of ECAI*, 359–363.

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. Satplan: Planning as satisfiability. In *Proceedings of IPC*.

Korf, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence* 26(1):35–77.

Newton, M. H.; Levine, J.; and Fox, M. 2005. Genetically evolved macro-actions in ai planning. In *Proceedings of PLANSIG*, 47–54.

Richter, S., and Westphal, M. 2008. The lama planner using landmark counting in heuristic search. In *Proceedings of IPC*.

Slaney, J., and Thiébaux, S. 2001. Blocks world revisited. *Artificial Intelligence* 125(1-2):119–153.

problem	SATPLAN			SGPLAN			LAMA		
	orig	ref no fr	ref with fr	orig	ref no fr	ref w fr	orig	ref no fr	ref with fr
depotprob1817	>600	>600	>600	24.69(100)	391.46(99)	0.15(95)	331.03(118)	95.94(111)	3.64(93)
depotprob1916	137.57(79)	49.27(74)	5.32(70)	0.41(83)	>600	80.95(57)	1.70(62)	0.58(59)	0.14(60)
depotprob4321	5.32(43)	2.24(46)	0.48(39)	0.02(41)	0.10(35)	0.00(34)	4.96(39)	3.69(38)	0.03(37)
depotprob4398	1.08(32)	0.72(36)	0.27(35)	0.04(28)	0.03(28)	0.00(28)	0.23(30)	0.10(27)	0.03(26)
depotprob5646	0.39(31)	0.26(28)	0.08(28)	0.01(26)	0.01(26)	0.00(28)	0.17(26)	0.07(26)	0.02(26)
depotprob5656	222.84(70)	76.73(71)	5.92(69)	411.33(133)	337.18(70)	0.24(62)	>600	>600	3.42(63)
depotprob6178	6.87(51)	4.54(51)	1.50(46)	0.11(48)	0.06(48)	0.02(37)	11.66(41)	6.06(41)	0.06(39)
depotprob6587	3.35(30)	2.18(29)	0.55(26)	0.06(28)	0.06(26)	0.00(24)	0.43(25)	0.19(25)	0.05(23)
depotprob7654	10.08(41)	6.13(39)	1.39(40)	0.09(35)	0.04(33)	0.01(33)	1.48(33)	46.58(35)	12.41(33)
depotprob8715	35.68(50)	28.31(46)	8.70(44)	0.26(34)	0.13(34)	0.05(36)	1.66(34)	0.54(33)	0.17(33)
dlog-2-3-6a	24.67(46)	3.52(43)	3.35(38)	0.94(41)	0.05(42)	0.02(42)	2.20(44)	1.68(49)	1.68(49)
dlog-2-3-6b	3.82(27)	1.31(27)	1.25(28)	0.01(32)	0.01(32)	0.01(32)	1.61(39)	1.03(30)	1.01(30)
dlog-3-3-6	2.86(37)	1.00(38)	0.90(35)	0.56(46)	0.14(41)	0.07(41)	0.15(51)	1.63(46)	1.63(46)
dlog-4-4-8	28.94(54)	13.08(52)	8.72(53)	0.04(47)	0.01(47)	0.01(47)	0.13(44)	0.06(44)	0.06(44)
dlog-5-5-10	>600	167.16(98)	169.36(95)	>600	25.20(103)	17.82(105)	18.24(132)	7.55(114)	7.43(114)
dlog-5-5-15	507.89(92)	110.19(84)	121.43(89)	6.93(110)	3.81(106)	1.63(106)	14.33(112)	6.20(95)	6.04(95)
dlog-5-5-20	>600	>600	406.60(92)	23.15(105)	>600	>600	17.75(127)	9.24(114)	8.88(114)
storage-11	85.47(22)	9.25(20)	-	0.00(17)	err	-	0.56(32)	0.08(20)	-
storage-12	38.01(24)	9.65(24)	-	0.00(17)	0.00(20)	-	0.70(32)	0.06(20)	-
storage-13	153.45(18)	64.26(18)	-	0.00(18)	0.00(20)	-	1.25(38)	0.02(20)	-
storage-14	24.70(22)	23.58(22)	-	0.00(19)	0.00(26)	-	5.15(32)	0.33(24)	-
storage-15	8.28(25)	0.60(26)	-	0.01(21)	0.00(20)	-	0.88(22)	0.56(20)	-
ztravel-3-7	1.90(22)	1.59(19)	-	0.01(18)	0.00(18)	-	0.04(15)	0.04(18)	-
ztravel-3-8a	1.71(27)	1.31(25)	-	0.01(27)	0.00(27)	-	0.05(24)	0.04(23)	-
ztravel-3-8b	0.86(27)	0.65(29)	-	0.01(29)	0.00(29)	-	0.04(28)	0.03(28)	-
ztravel-3-10	3.79(31)	3.78(38)	-	0.02(36)	0.01(36)	-	0.05(31)	0.05(30)	-
ztravel-5-10	34.49(42)	22.53(38)	-	0.22(40)	0.19(40)	-	0.24(41)	0.20(39)	-
ztravel-5-15a	92.04(50)	40.70(53)	-	0.12(60)	0.09(60)	-	0.48(47)	0.37(47)	-
ztravel-5-15b	err	err	-	0.58(55)	0.47(55)	-	0.62(57)	0.48(57)	-
gold-miner-7x7-01	5.99(35)	4.97(35)	-	err	0.00(33)	-	0.30(176)	0.04(31)	-
gold-miner-7x7-02	4.36(32)	3.58(32)	-	err	0.00(30)	-	0.16(161)	0.04(28)	-
gold-miner-7x7-03	4.12(32)	3.46(32)	-	err	0.01(34)	-	0.29(257)	0.04(32)	-
gold-miner-7x7-04	9.15(43)	7.62(42)	-	err	0.01(41)	-	0.21(130)	0.02(41)	-
gold-miner-7x7-05	9.78(39)	8.16(41)	-	err	0.00(39)	-	0.38(157)	0.03(39)	-
gold-miner-7x7-06	6.00(33)	4.96(34)	-	err	0.00(33)	-	0.18(182)	0.02(33)	-
gold-miner-7x7-07	6.08(38)	4.85(38)	-	err	0.01(36)	-	0.04(65)	0.02(34)	-
gold-miner-7x7-08	3.06(25)	2.60(25)	-	err	0.00(27)	-	>600	0.02(25)	-
gold-miner-7x7-09	4.24(33)	3.61(29)	-	err	0.01(31)	-	0.14(130)	0.03(29)	-
gold-miner-7x7-10	5.96(35)	4.92(35)	-	err	0.00(33)	-	0.29(176)	0.05(31)	-
matching-bw-n15a	27.81(42)	19.93(42)	1.95(42)	>600	>600	0.25(46)	0.22(68)	0.22(62)	0.25(50)
matching-bw-n15b	34.25(56)	10.18(52)	1.37(52)	>600	>600	170.39(54)	0.36(66)	0.31(78)	0.38(60)
matching-bw-n15c	26.80(38)	9.39(42)	1.20(38)	>600	20.89(74)	284.65(34)	0.71(66)	1.74(72)	0.14(46)
matching-bw-n15d	40.74(40)	14.41(42)	1.45(42)	>600	>600	>600	0.51(58)	0.29(60)	0.18(56)
matching-bw-n15e	59.00(34)	14.62(36)	1.73(36)	>600	>600	47.17(42)	0.17(40)	0.80(44)	0.07(34)
matching-bw-n20a	>600	189.75(52)	15.00(50)	>600	>600	>600	0.46(62)	0.37(66)	0.25(66)
matching-bw-n20b	245.12(48)	54.35(50)	4.39(48)	>600	>600	0.35(60)	3.63(78)	3.44(68)	0.93(50)
matching-bw-n20c	363.36(54)	60.94(54)	5.62(54)	>600	533.89(90)	237.64(36)	>600	55.72(70)	1.46(72)
matching-bw-n20d	195.87(46)	43.04(46)	4.31(46)	>600	10.40(74)	>600	0.77(86)	1.43(92)	0.48(64)
parking-a	>600	399.11(16)	-	0.67(20)	0.02(31)	-	0.16(19)	0.15(31)	-
parking-b	>600	98.13(15)	-	0.73(22)	0.11(36)	-	0.22(20)	0.14(31)	-
parking-c	304.47(12)	17.68(12)	-	0.53(25)	0.08(29)	-	0.16(15)	0.12(23)	-
parking-d	>600	>600	-	0.02(18)	0.01(18)	-	0.34(31)	0.13(18)	-
parking-e	>600	167.20(13)	-	0.76(29)	0.05(56)	-	0.31(27)	0.14(20)	-
parking-f	>600	>600	-	0.47(25)	0.27(39)	-	0.20(21)	0.13(19)	-
parking-g	>600	>600	-	17.89(34)	0.83(52)	-	14.16(30)	0.58(20)	-
parking-h	>600	>600	-	19.39(37)	0.71(58)	-	0.55(19)	1.01(38)	-
thoughtful-s5-t4d	3.20(37)	0.47(37)	-	0.04(28)	0.02(30)	-	err	err	-
thoughtful-s5-t4e	44.41(41)	2.56(36)	-	0.05(33)	0.02(31)	-	err	err	-
thoughtful-s5-t4f	5.46(36)	0.94(32)	-	0.02(31)	0.02(28)	-	err	err	-
thoughtful-s5-t4g	3.92(38)	1.11(38)	-	0.03(32)	0.02(32)	-	err	err	-
thoughtful-s5-t4h	3.26(41)	0.58(36)	-	0.04(31)	0.01(34)	-	err	err	-
thoughtful-s7-t5a	288.57(58)	99.73(65)	-	0.10(50)	205.71(49)	-	err	err	-
thoughtful-s7-t5b	136.44(65)	unsolvable	-	1.39(84)	unsolvable	-	err	unsolvable	-
thoughtful-s7-t5c	232.32(71)	unsolvable	-	0.13(51)	unsolvable	-	err	unsolvable	-

Table 2: The table shows comparison of the running times (in seconds) and plan lengths (in brackets) of original problems and problems reformulated using our methods (without and with ‘flaws‘ ratio).