

# Transfer of Domain Knowledge in Plan Generation

## Learning Goal-dependent Annulling Conditions for Actions

Michael Siebers

Received: date / Accepted: date

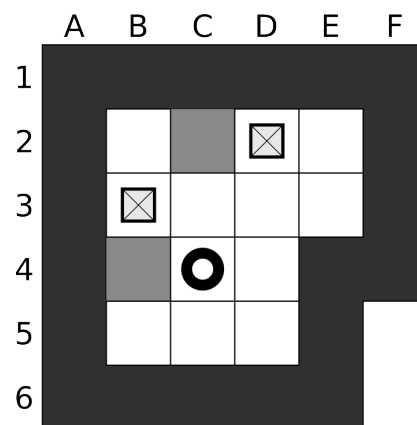
**Abstract** In this paper we present an approach to avoid dead-ends during automated plan generation. A first-order logic formula can be learned that holds in a state if the application of a specific action will lead to a dead-end. Starting from small problems within a problem domain examples of states where the application of the action will lead to a dead-end will be collected. The states will be generalized using inductive logic programming to a first-order logic formula. We will show how different notions of goal-dependence could be integrated in this approach.

The formula learned will be used to speed-up automated plan generation. Furthermore, it provides insight into the planning domain under consideration.

**Keywords** automated planning · learning domain knowledge · dead-end states · inductive logic programming

### 1 Motivation

Automated planning is one of the fundamentals of artificial intelligence. Using an adequate representation for states in the world and a pool of actions to modify the current state modern programs are able to address lots of problems. A lot of effort was put into developing different planning algorithms and adequate heuristics guiding the search for a good plan. There is a competition at irregular intervals (International Planning Competition, IPC) at the International Conference on Auto-



**Fig. 1** A sample Sokoban board state. The circle denotes the movable player token. The small squares at B3 and D2 represent boxes. The darker squares (B4 and C2) denote goal positions. The board is completely surrounded by walls (dark grey).

ated Planning and Scheduling where leading planning teams are able to compare their respective planners. For this competition a library of problems in a standardized language (Planning Domain Description Language, PDDL) is available. These problems are quite complex. However, automated planning suffers one major shortcoming: The planning process starts from scratch for any given problem. Even if the problem description of a solved problem is only slightly altered the present plan is discarded completely. No knowledge is gained or transferred. In this regard humans are still superior to automated planners.

Consider for instance the single-player full-knowledge game Sokoban (see Fig. 1): The aim of the game is to move boxes onto given goal positions. The game board consists of squares containing either a wall, a box or the player token. There are as many squares marked as

M. Siebers  
Cognitive Systems Group  
Faculty Information Systems and Applied Computer Sciences  
University of Bamberg, Germany  
Tel.: +49-951-863 2863  
E-mail: michael.siebers@uni-bamberg.de

goal positions as there are boxes in the game. No goal square may contain a wall. However, it may contain a box or the player token. In every round the player token may be moved vertically or horizontally onto an adjacent square. If the target square contains a box it is pushed in the moving direction. A box may only be pushed onto an empty square. Moving more than one box at the same time is not possible. An example board state is given in Figure 1.

A box in a corner square, that is a square with walls adjacent in horizontal and vertical direction, can never be pushed. So pushing a box onto a corner square (for instance E2 in the example board given) is only sensible if the corner square is marked as goal position. Any human player will gain this knowledge in short time. If not by insight then after erroneously pushing a box on a corner square for the first time.

The aim of our project is to mimic this human competence. We want to induce knowledge by analyzing easy problems and transfer this knowledge to more general problems within the same problem domain.

In the next section we will give some basic information about automated classical planning. In Section 3 we will consider related approaches on learning for automated planning. In Section 4 we will detail how we are going to learn and transfer the knowledge sought. We will conclude by giving our next steps in Section 5.

## 2 Classical Planning

The aim of classical planning is to find a path from the known initial state of a problem to some state satisfying the goal condition. In STRIPS (Fikes and Nilsson, 1971) representation states are formalized as sets of true positive literals defined over a set of predicate symbols  $P$  and a given set of objects  $O$ . The transitions—called actions—are represented by the *precondition* list, the *add-list*, and the *del-list*. Some action  $a$  is applicable in a state  $s$  if all literals in the precondition ( $\text{pre}_a$ ) are true in this state. In the successor state  $s'$  the literals in the add-list ( $\text{Add}_a$ ) are true and the literals in the del-list ( $\text{Del}_a$ ) are false.

$$s \xrightarrow{a} s' \Leftrightarrow [\text{pre}_a \subset s \wedge s' = (s \cup \text{Add}_a) \cap \text{Del}_a]$$

Multiple planning problems are grouped into domains. A domain  $\mathcal{D} = \langle P, A \rangle$  consists of the predicate symbols necessary to formulate the problem and the set of actions available to solve the problem. Thus a planning problem  $\mathcal{P}_{\mathcal{D}} = \langle O, s_0, g \rangle$  is fully defined by the available objects  $O$ , the initial state  $s_0$ , and a goal condition  $g$  given as a set of positive literals. A state  $s$  is called goal state if all literals in  $g$  are true ( $g \subseteq s$ ).

Extensions to this formulation (e. g., non-boolean variables, negation) are often convenient but not essential.

In the *planning as search* formulation, a problem  $\mathcal{P}_{\mathcal{D}}$  is solved by path-finding algorithms over the state space graph  $G(\mathcal{P}_{\mathcal{D}})$  of the problem. The nodes of the state space graph represent the states in the model. The edges  $(s, s')$  correspond to actions. Since the number of possible states grows exponentially with number of predicates and objects, even small problems have huge state spaces. Thus, the state space graph is not build explicitly. Furthermore, heuristic search is used.

In some way classical planning resembles reinforcement learning (RL). Both seek the best action to apply in a given state. However, RL aims at finding such assignments for every state in the state space, whereas classical planning only aims at those on the path from the given starting state to some goal state. Furthermore, RL uses trial-and-error to determine the actions maximizing some reward, whereas planning relies on the analysis of the problem at hand and graph search.

## 3 Related work

To our knowledge we are the first to learn under which conditions actions lead to dead-ends. However, we are not the first to learn to guide plan generation.

The planning architecture Prodigy (Minton et al, 1989) includes machine learning approaches to learn plan generation guiding rules (Minton, 1988). In his SCOPE-system Estlin (1998) learns domain-specific if-then-rules for action selection using ILP and explanation based learning. The selection of actions may also depend on the current goal.

In a more recent approach de la Rosa and McIlraith (2011) learn first-order linear temporal logic formulas to guide plan generation. They learn conditions over state transitions. The learned formulas are used to prune the search space. They also use a notation of goal dependence. However, they only support the naïve approach to goal dependence described in the previous section. The learned formulas are specifically designed to be used with TLPlan (Bacchus and Kabanza, 2000).

Martín and Geffner (2004) learn functions which map states and goals into actions. This function is based on concepts-languages which have the expressive power of subsets of first-order logic. However, like SCOPE, this approach does not provide insight in a domain.

## 4 Conditional Annuling of Actions

As explained in Section 1 we aim to learn which actions to avoid in certain situations. In fact we want

to learn the conditions under which the application of an action  $a$  will lead to a *dead-end*. A state is called a dead-end if there exists no path from this state to any goal state. Thus, if the annulment condition  $\phi_a$  for action  $a$  holds in some state the action must not be applied. We will learn these annulment conditions from small problems<sup>1</sup> and transfer the generalized knowledge to the complete domain.

Since states are represented by sets of literals we will formalize the annulment conditions as function-free first order logic formulas defined over the set of predicate symbols  $P$ . We will use inductive logic programming (ILP) to abstract the states to the formula  $\phi_a$ . In the next sections we will show how this is done. Furthermore, we will show how the states to abstract from are gathered and how the learned formulas can be used during planning.

#### 4.1 Learning Annulment Conditions

As the set of objects is specific for a problem but not for a domain the annulment condition may not contain any objects.<sup>2</sup> Thus, we want to induce a function-free formula that contains no objects and no unbound variables. If we restrict this to a formula which contains only existential quantifiers this is equivalent to horn clauses. To learn these clauses any inductive logic programming algorithm—like FOIL (Quinlan, 1990) or GOLEM (Muggleton and Feng, 1992)—can be used.

In order to induce such a set of horn clauses we need background knowledge and positive and negative example sets. We assume that we have a set of states  $S_a^+$  where the application of  $a$  leads to a dead-end and a set of states  $S_a^-$  where this is not the case. How these sets of states are constructed will be explained in the next section. Let the predicate to be learned be called  $\text{annul}_a$ . We now can form the set of positive examples  $F_a^+ = \{\text{annul}_a(s) | s \in S_a^+\}$  and the set of negative examples  $F_a^- = \{\neg \text{annul}_a(s) | s \in S_a^-\}$ . The background knowledge  $B$  must contain everything we know about the states in  $S_a^+$  and  $S_a^-$ . Though states are sets of literals, this cannot be the union of the states in both sets. The literals in each state are only true within the state. So the literals are implicitly dependent on the state. To use the knowledge this dependence must be made explicit. Every literal  $p(t_1, \dots, t_n)$  in state  $s_i$  must be

reformulated such that the state—in fact some unique id—is part of the predicate  $p_s(t_1, \dots, t_n, s_i)$ . The union of the resulting sets is then the background knowledge.

Using any ILP algorithm we can use  $F_a^+$ ,  $F_a^-$ , and  $B$  to infer a set of clauses  $C_a$  that explicitly states in which states to annul  $a$ .

If we want to allow universal quantifiers in the formula we must rewrite the background knowledge. Since all objects of a planning problem are known we can check for each state  $s_i$  if some predicate  $p$  holds for all possible arguments. If this is the case we can add the literal  $p_{\text{all}}(s_i)$  to the background knowledge. Using this extended background knowledge simple universal quantified formulas are possible. Predicates for more sophisticated formulas can be constructed accordingly.

#### 4.2 Using the Knowledge

Having the set of clauses  $C_a = \{c_1, \dots, c_m\}$  for an action  $a$  we can use it during planning. As first step we construct the function-free formula  $\phi_a$  from the set of clauses. Since, this formula is to be applied on states we make the state dependence of the clauses implicit reversing the modification from  $p$  to  $p_s$ . This results in clauses of the form  $\neg(l_1 \wedge \dots \wedge l_n) \vee \text{annul}_a$  implicitly universal quantified over all variables occurring in the literals. This is equivalent to the formula  $l_1 \wedge \dots \wedge l_n \rightarrow \text{annul}_a$ , implicitly *existential* quantified over all variables occurring in the literals. Revoking the universal quantification predicates  $p_{\text{all}}$  to explicit universal quantifications we get a set of formulas defining annulment conditions for the action  $a$ . This set can be merged to a single formula  $\phi_a$  or-ing the elements' premises.

If we allow arbitrary formulas as action preconditions in our planning domain representation we can add  $\neg\phi_a$  to the preconditions of action  $a$ . Thus, an action is not applicable if the annulment condition holds and dead-ends are avoided during planning. If only positive literals are allowed as preconditions additional predicates modelling the existential and universal quantified expressions must be introduced. The actions must be split and modified such that these additional predicates are taken into account. Since most modern planning systems support existential and universal quantification we will not detail these modifications.

#### 4.3 Finding the State Sets

As mentioned before we need a set of positive and a set of negative example states. Since we are dealing with simple problems the naïve way to obtain those is an exhaustive search. Every path from the starting state

<sup>1</sup> We consider problems as small if their number of states is low.

<sup>2</sup> The domain specification may be augmented with a set of constants. Those constants represent objects which are present in every problem of this domain. Since there are ILP algorithms (e. g. FOIL) that can induce formulas with specific objects, this augmentation poses no problem to our approach.

is expanded. If no actions are applicable in any non-goal-state the state is a dead-end. If all children of a state are dead-ends the state itself is also a dead-end. All non-dead-end states where applying action  $a$  does lead to a dead-end form the set of positive examples  $S_a^+$  for this action, all non-dead-end states (where  $a$  is applicable) form the set of negative examples  $S_a^-$ .

However, this approach is only feasible for the smallest of problems. Larger problems can be handled by using heuristic search counter-intuitively: During path expansion it is first checked whether its final state is a dead-end. If so, the next-to-last state is a positive example for annulling the last action of the path. Otherwise, the next-to-last state is a negative example. Dead-ends are not further expanded. Additionally, paths with a *high* heuristic value could be preferred for expansion. This way the probability of dead-end occurrence might be increased. Since the search is no more exhaustive, the identification of dead-ends is still an issue.

#### 4.4 Incorporating Goal-Dependence

As we have seen in the Sokoban example in Section 2 the annullment of actions is not only dependent on the current state and the action, but it also depends on the goal definition. Thus, this information must be included during learning and exploited during application. So our examples for an action are not only the states, but they are 2-tuples  $\langle s, d \rangle$  of state  $s$  and goal description  $d$ .

The naïve choice for the goal description is the goal condition  $g$  of the problem. Using the situation calculus trick introduced above every literal  $p(t_1, \dots, t_n)$  in  $g$  is extended and renamed to  $p_g(t_1, \dots, t_n, s_i)$ . These literals can be used as additional background knowledge during learning. The learned  $\phi_a$ 's may then contain goal dependent predicates. These can be pre-evaluated for a given problem and thus be used during application.

However, using the goal condition might be too restrictive. There might be literals which hold as soon as a goal state is reached but which are not part of the goal condition. In the Sokoban example this might be the fact that the player token is at D2. Furthermore, there are literals which must be true to be able to reach a goal state. In the Sokoban example one example is the fact that the player token is at D3. This two kinds of goal-dependence might also play an important role when deciding the annullment of actions. The second approach might be solved using landmarks (Hoffmann et al, 2004). Landmarks are sets of literals such that at least one literal of each set must hold once during each valid plan. The first condition type is still an open problem.

## 5 Conclusion and Next Steps

We presented an approach to learn to avoid dead-ends. We learn first-order logic formulas which hold if applying an action leads to a dead-end. These formulas can be used as additional preconditions for actions and thus serve to prune the search space. Furthermore, these formulas may provide insight in planning domains.

The next step is to implement the system and to test it on different domains. These will be domains which are known to contain dead-ends (e.g., **rovers** or **airport** from IPC 2011) and dead-end free domains. The results will be evaluated for plan generation speed-up. Further work will include the types of goal-dependence and the use of heuristics for example extraction.

## References

- Bacchus F, Kabanza F (2000) Using temporal logics to express search control knowledge for planning. *Artificial Intelligence* 116(12):123–191
- Estlin TA (1998) Using multi-strategy learning to improve planning efficiency and quality. PhD thesis, Artificial Intelligence Laboratory, The University of Texas at Austin
- Fikes RE, Nilsson NJ (1971) Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4):189–208
- Hoffmann J, Porteous J, Sebastia L (2004) Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22:215–278
- Martín M, Geffner H (2004) Learning generalized policies from planning examples using concept languages. *Applied Intelligence* 20(1):9–19
- Minton S (1988) Learning effective search control knowledge: an explanation-based approach. Kluwer Academic Publishers
- Minton S, Knoblock C, Kuokka D, Gil Y, Joseph R, Carbonell J (1989) Prodigy2.0: the manual and tutorial. Tech. rep., Carnegie Mellon
- Muggleton S, Feng C (1992) Efficient induction in logic programs. In: Muggleton S (ed) *Inductive Logic Programming*, Academic Press, pp 281–298
- Quinlan J (1990) Learning logical definitions from relations. *Machine Learning* 5(3):239–266
- de la Rosa T, McIlraith SA (2011) Learning domain control knowledge for TLPlan and beyond. In: *Proceedings of the ICAPS-11 Workshop on Planning and Learning (PAL)*