Nogood Learning for Classical Planning

Work in progress report

Igor Razgon
Computer Science Department Ben-Gurion University of the Negev
Beer-Sheva, 84-105, Israel
irazgon@cs.bgu.ac.il

Introduction

Methods of constraint satisfaction and SAT have been successfully used in the planning area. Examples are translation of planning instances into propositional SAT formulas (Kautz, McAllester, & Selman 1996; Kautz & Selman 1996; 1998), applying dynamic constraint satisfaction methods to Graphplan structures (Kambhampati 2000), compilation of Graphplan into a constraint satisfaction problem (CSP) instance (Do & Kambhampati 2001). In this paper we propose a method of pruning search space based on *nogood learning*.

Techniques of nogood learning are used in constraint satisfaction (Dechter 1986; Ginsberg 1993). The essence of nogood learning is maintaining a set of partial assignments (nogoods) that cannot be extended to a full solution. The constraint solver discovers nogoods during its work. Thus nogoods are constraints acquired during search. It is shown theoretically and empirically that maintaining a set of nogoods of a polynomial size can improve performance of a constraint solver (Kondrak & van Beek 1995; Bayardo & Miranker 1996).

The method proposed in the paper is applied to a forward chaining planner (Bacchus & Teh 1998) acting in a space of states. Every state is a subset of propositions or atoms of the universe. A plan is a sequence of states. The transition between adjacent states is performed by an action of the given set of actions. The task is to find a plan whose first state is the initial state, and the last state contains the goal. We assume that the planner has a parameter determining the maximal length of the produced plan. Actions are represented in the STRIPS-like style (Fikes & Nilsson 1971) by lists of their preconditions, add, and delete effects.

A forward chaining planner constantly holds *the current plan* starting at the initial state. At every iteration, the planner tries to extend the current plan. That is, the planner tries to append to the current plan a state immediately reachable from the *current state* (the last state of the current plan). If the planner detects that the current plan cannot be a prefix of any full plan, it backtracks. We call states immediately reachable from the current state *extensions* of the current plan.

We assume that the following three conditions can cause the planner to backtrack (Kambhampati, Katukam, & Qu 1996).

- Every extension of the current plan is a member of the plan.
- A method approximating the distance from the current state to the goal (Kambhampati & Nigenda 2000; Bonet & Geffner 2001) detects that the current plan can not be a prefix of a full plan of length less than or equal to the maximal length.
- Every extension of the current plan eventually results in backtrack.

Let us call the first two conditions *the basic backtrack conditions*, the last one *the complex backtrack condition.* The complex backtrack condition can be reformulated in a non-recursive way as follows: *every sequence of extensions of the current plan eventually meets a basic backtrack condition.* Unfortunately, to detect the complex backtrack condition in such a way, the planner should consider all "dead-ends", the number of which can be exponentially large.

We propose a method that allows detection of the complex backtrack condition without actual reaching all dead-ends. Whenever backtrack is performed, the planner constructs a *nogood* of the current plan. This nogood is a pattern explaining the property that forces the plan to be discarded. The discovered patterns are maintained in a *database of nogoods.* Using the database, one can formulate yet another backtrack condition: *The current plan satisfies some pattern in the database.* Let us call this condition *the explanation based backtrack condition.* It is the complex backtrack condition discovered in a non-recursive way.

We describe a formal language of representation of nogoods. In particular, we show encoding of the two basic backtrack conditions and a method of construction of complex nogoods. Because the number of different patterns can be exponentially large, we provide a heuristic estimation of patterns. The pattern with the lowest estimation is deleted from the database in the case of "overflow".

The present paper is closely related to (Kambhampati, Katukam, & Qu 1996) where a failure driven pruning method for a partial order planner was presented. Another failure-driven pruning method was described in (Bhatnagar & Mostow 1994). The method is applied to a state-based planner. Unlike the algorithm proposed in the present paper, the method provided in (Bhatnagar & Mostow 1994) affects the search heuristic and can lead to incompleteness.

The algorithm proposed in the present paper, the methods presented in (Kambhampati, Katukam, & Qu 1996; Bhatnagar & Mostow 1994), and the nogood learning algorithms for constraint satisfaction can be considered as examples of explanation based learning techniques (Minton *et al.* 1989).

The rest of the paper is organized as follows. Section 2 presents a method of nogood learning during planning. Section 3 describes a context in which the method can be useful.

Learning of Nogoods During Search

Construction of Nogoods for Basic Backtrack Conditions

Let P be the current plan at some moment of execution of a forward chaining planner. Let $Last(P)$ be the last state in the plan (as it was said above, we consider plans as sequences of states). Let $S = \{s_1, s_2, \ldots s_m\}$ be the set of states reachable from $Last(P)$ by one of actions of the given planning domain (in other words $Last(P)$ is the set of possible extensions of the current plan).

Given the notation, one can formulate the first basic backtrack condition as follows.

For every i, $1 \leq i \leq m$, $s_i \in P$

The nogood constructed for this case is of the form $\langle Last(P), S\rangle$. This means that backtrack should be performed when the last state of the current plan is $Last(P)$ and all states of S are members of the current plan.

Let $Size(P)$ be the size of plan P and $MaxSize$ be the maximal possible size of a plan. Assume that a distance approximation method (Bonet & Geffner 2001; Kambhampati & Nigenda 2000) applied to $Last(P)$, detects that it is impossible to reach a state containing the goal performing less than X steps [1], $X < MaxSize - Size(P)$. This is the second basic backtrack condition. A nogood appropriate for this case is $\langle Last(P), MaxSize - X\rangle$. This nogood means that the state $Last(P)$ cannot be the last in a plan longer than $MaxSize - X$ steps.

The General Format of Nogoods

The general format of nogoods is $\langle st, S, L\rangle$, where st is a state, S is a set of states, L is an integer number. This nogood means that st cannot be the last state of a plan longer than L if all states of S are members of this plan.

[1] We assume that the distance approximation used is admissible and never overestimates the distance

Definition 1 *Let* $< st, S, L >$ *be a nogood, and let* P *be a plan. If* $st = Last(P)$, $Size(P) > L$ *and every state of* S *is a member of* P, *then* P **satisfies** *nogood* $< st, S, L >$.

Note that nogoods constructed for the first and second basic backtrack conditions do not contain the third and the second field respectively. We can extend them to the general format placing 0 in the third field of the nogood constructed for the first basic backtrack condition, and $\emptyset$ in the second field of the nogood constructed for the second basic backtrack condition.

Maintaining Database of Nogoods During Planning

The proposed version of forward chaining maintains a database of nogoods during its execution. Initially this database is empty. The algorithm performs the following two types of additional operations:

- Every time when a backtrack condition is satisfied, the appropriate nogood is generated and inserted into the database of nogoods
- Every time when a new current plan is generated, the planner checks whether the current plan satisfies some nogood maintained in the database.

In the case of complex backtrack condition, when the planner rejects every extension of the current plan P, the nogood of the current plan is constructed as follows.

Let $\langle st_1, S_1, L_1\rangle, \ldots \langle st_m, S_m, L_m\rangle$ be the set of nogoods satisfied by the extensions of P. Then the nogood formulated for P is $\langle Last(P), \bigcup_{i=1,m} S_i, (Max_{i=1,m} L_i) - 1\rangle$

Theorem 1 *Maintaining the database of nogoods in the form shown above, the planner does not loses solutions. Moreover, it does not increase the number of partial plans visited.*

Restriction the Size of the Database of Nogoods

The number of nogoods in the database of nogoods can be exponentially large. In this section we show an algorithm for restriction of the database of nogoods. We divide all nogoods into two types: *relevant* and *optional* ones.

Relevant nogoods are those that satisfy extensions of prefixes of the current plan. If n is the size of the current plan and m is the maximal number of states immediately reachable from a state of the current plan, then the number of such nogoods is $O(n * m)$. These nogoods are the necessary part of the database. All other nogoods are optional. Let $MaxOptSize$ be the maximal possible number of optional nogoods in the database, which is a parameter of our planner. If the number of optional nogoods in the database exceeds $MaxOptSize$, the planner has to decide which nogood has to be deleted from the database.

We propose to estimate nogoods by difficulty of their derivation. In the case when the number of optional

nogoods in the database is more than the allowed, the planner deletes the nogood with the lowest value of difficulty.

When a new nogood is constructed, its difficulty is estimated as follows:

- If the nogood is constructed as a result of satisfaction of a basic backtrack condition, its difficuly is set to 1
- If the nogood is constructed as a result of satisfaction of the complex backtrack condition, its difficulty is set to 1 plus the sum of the estimations of the nogoods of the extensions of the current plan.

Note, that according to this estimation, the difficulty of a nogood is an approximation of the number of backtracks required to derive it.

Applications of the Nogood Learning

The method presented in the paper performs pruning of a search space of a forward chaining planner. It was pointed out in (Bacchus & Teh 1998) that pruning of a search space has great impact on efficiency of forward chaining planners. In this section we describe a particular situation where the method can be worthwhile.

Maintaining of a database of nogoods can be useful when a planner is applied to a domain where its search heuristic does not help. An example is a planner, based on ordering of propositions of the goal (Koehler & Hoffmann 2000; Porteous & Sebastia 2000; Razgon & Brafman 2001), applied to a "puzzle" domain, in which already achieved propositions of the goal sometimes have to be destructed to achieve the rest (non-serializable goals in the terms of Korf (Korf 1985)). Examples of domains of this type are $n^2 - 1$-puzzle, Hungarian cube, Freecell, etc.

When planners are applied to an "inconvenient" domain, they perform exhaustive search. Maintaining a database of nogoods can help them to prune a significant fraction of their search space.

Further Development

In this paper we presented research results concerning use of nogood learning for classical planning. In particular, we described a format of representation of nogoods, we proposed an algorithm for construction of complex nogoods and a method for estimation of nogoods. The last method allows deletion from "overflowed" database the nogood with the lowest estimation.

The next stage of the research is to find a way of implementation making the gain of use the database of nogoods more than the expenses of its maintenance.

We suggest implementation of the planner as two parallel processes cooperating in the form of "master-slave". The master is the forward chaining planner. The slave is the process maintaining the database of nogoods.

The master cooperating with the slave can perform the following actions:

1. Every time when the master generates a new plan, it sends to the slave a request to check whether the plan satisfies some nogood in the database
2. If the master receives from the slave a message that some plan satisfies a nogood in the database, and this plan is a prefix of the current plan, the master "backjumps", to the last state of the received plan
3. If the master performs backtrack itself witout a message from the slave, it sends to the slave a request to generate a nogood for the discarded current plan.

The slave answers to the requests of the master. If it has a "free time" when there is no request from the master, it can perform additional reasoning of the database of nogoods, increasing the pruning effect.

The main expected effect of the model of cooperation is that the master does not waste time to unsuccessful checks. Really, assume that to check if the current plan satisfies some nogood in the database takes time t. In the proposed model, the master sends the checking requests to the slave and proceeds to work. In the case when the current plan does not satisfy any nogood in the database, the master spends no time to checking! If the current plan does satisfy a nogood, the checking time is the same as in the sequential model plus the time of message transfer.

References

Bacchus, F., and Teh, Y. W. 1998. Making forward chaining relevant. In *International Conference on Artificial Intelligence Planning Systems*, volume 4, 54–61. AAAI Press.

Bayardo, R., and Miranker, D. 1996. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96), Volume 1*, 298–304. AAAI Press / The MIT Press.

Bhatnagar, N., and Mostow, J. 1994. line learning from search failures machine learning. *Machine Learning* 15:69–117.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Dechter, R. 1986. Learning while searching in constraint-satisfaction problems. In *National Conference on Artificial Intelligence*, 178–183.

Do, M. B., and Kambhampati, S. 2001. Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence* 132(2):151–182.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving and problem solving. *Artificial Intelligence* 2:189–208.

Ginsberg, M. L. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Kambhampati, S., and Nigenda, R. S. 2000. Distance-based goal-ordering heuristics for graphplan. In *Artificial Intelligence Planning Systems*, 315–322.

Kambhampati, S.; Katukam, S.; and Qu, Y. 1996. Failure driven dynamic search control for partial order planners: An explanation based approach. *Artificial Intelligence* 88(1-2):253–315.

Kambhampati, S. 2000. Planning graph as a (dynamic) csp: Exploiting ebl, ddb and other csp search techniques in graphplan. *Journal of Artificial Intelligence Research* 12:1–34.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In Shrobe, H., and Senator, T., eds., *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, 1194–1201. Menlo Park, California: AAAI Press.

Kautz, H., and Selman, B. 1998. Blackbox: A new approach to the application of theorem proving to problem solving.

Kautz, H. A.; McAllester, D.; and Selman, B. 1996. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, 374–384.

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

Kondrak, G., and van Beek, P. 1995. A theoretical evaluation of selected backtracking algorithms. In Mellish, C., ed., *IJCAI'95: Proceedings International Joint Conference on Artificial Intelligence.*

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

Minton, S.; Carbonell, J.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

Porteous, J., and Sebastia, L. 2000. Extracting and ordering landmarks for planning.

Razgon, I., and Brafman, R. 2001. A forward search planning algorithm with a goal ordering heuristic. In *6th European Conference on Planning.*