Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning

Heuristic Guided Optimization for Propositional Planning

Andreas Sideris and **Yannis Dimopoulos**
Department of Computer Science
University of Cyprus
Nicosia, Cyprus

Abstract

Planning as Satisfiability is an important approach to Propositional Planning. A serious drawback of the method is its limited scalability, as the instances that arise from large planning problems are often too hard for modern SAT solvers.

This work tackles this problem by combining two powerful techniques that aim at decomposing a planning problem into smaller subproblems, so that the satisfiability instances that need to be solved do not grow prohibitively large. The first technique, *incremental goal achievement*, turns planning into a series of boolean optimization problems, each seeking to maximize the number of goals that are achieved within a limited planning horizon. This is coupled with a second technique, called *heuristic guidance*, that directs search towards a state which satisfies all goals.

Introduction

Planning as Satisfiability (Kautz and Selman 1996; Kautz, Selman, and Hoffmann 2006) is an important method for optimal propositional planning. One of the advantages of the method is that it finds plans of optimal makespan. However, it often happens that the instances generated from planning problems, grow so large that they are unsolvable even by the most advanced SAT solvers.

This motivated a number of recent attempts to improve scalability at the cost of sacrificing the optimality of the generated solutions. One such example is `Madagascar` (Rintanen 2012), that enhanced a SAT solver with a planning specific variable selection heuristic, to generate suboptimal plans quickly. In a different spirit, the `PSP` system (Sideris and Dimopoulos 2012), introduced the idea of *planning as optimization*. `PSP` employs the increasing planning horizon technique, but unlike classic `SATPLAN`, it maximizes the goals that are achieved within each horizon, and adds them as intermediate goals in the problem model. However, as the plan length can grow arbitrarily long, at some point it is expected that both approaches will be confronted with performance difficulties.

In this work we take a different path. The basic idea is to split a planning problem into smaller subproblems, of a size that is not prohibitive for the underlying solver. `PSP-H`, the new planning system that is described here, shares with `PSP` the planning as optimization perspective, but it differs in a number of important ways. The first is the *incremental goal achievement* which, at a high level, works as follows. `PSP-H` first generates a sub-plan that, starting from the initial state, achieves a subset, of predefined size, of the problem goals. The state that results after the execution of the actions of this sub-plan, becomes the new initial state, and a new sub-plan that satisfies a larger subset of goals is computed. The procedure iterates and links together the subplans that are generated along the way. Therefore, instead of solving the original planning problem, `PSP-H` solves a series of smaller subproblems.

A downside of this greedy approach is that it focuses on maximizing the number of achieved goals in a limited planning horizon, and it ignores completely goals that cannot be achieved within this horizon. In order to overcome the limitations that this would place on the effectiveness of the system, `PSP-H` is enhanced by a second technique called *heuristic guidance*, which imposes an additional requirement on the intermediate states that are computed by the `PSP-H` algorithm. The property that these states need to satisfy is that there must be a *relaxed* plan from each such state to the final state. `PSP-H` employs three different relaxation methods that are all based on ignoring some of the problem constraints, but they differ in their strength.

The first relaxation method is the well-known delete lists relaxation, whereas the second method ignores all action mutexes. The last relaxation, which is stronger than the first but weaker than the second, ignores action mutexes and uses a subset of the fact mutexes that are heuristically selected.

`PSP-H` is a incomplete and suboptimal planner implemented on top of the `PSP` system. Our experimental evaluation on a number of domains taken from planning competitions, demonstrates that `PSP-H` can solve challenging problems, that require long plans. Moreover, a preliminary comparison with `Madagascar` shows that the new system can solve more problems in some domains, and generate better quality solutions both in term of plan length and number of actions.

Some of the techniques employed in `PSP-H` are similar to those used in other planning systems. For instance, the local optimization method of `PSP-H` bares some resemblance to the enforced hill-climbing approach of the

FF planner (Hoffmann and Nebel 2001). The CPT system (Vidal and Geffner 2006) combines heuristic estimators with constraint programming techniques in the context of optimal temporal planning. Finally, a study of the relation between the relaxation methods of PSP−H and the red-black heuristics of (Katz, Hoffmann, and Domshlak 2013a; 2013b) may lead to a tighter integration of heuristic and constrained optimization based planning.

Background

We assume familiarity with the planning as satisfiability framework. We discuss in brief pseudo-boolean optimization and the planning as optimization method of PSP.

A (STRIPS) planning problem is a triple $P =< I, G, \mathcal{A} >$, where I is the set of facts that hold in the initial state, G are the goals, and $\mathcal{A}$ is a set of actions. Each action has preconditions, add effects, and delete effects. In the SAT model of a planning problem, time-stamped propositional atoms (or variables) represent the action and facts of the problem. An atom $A(T)$, where A is an action, corresponds to the decision of whether action A is taken or not at time T, and analogously for variables of the form $f(T)$, where f is a fact.

PSP−H uses the SMP (Sideris and Dimopoulos 2010) encodings of a planning problem, that derives its clauses from the *planning graph* (Blum and Furst 1997) of the problem. Among the clauses of the SMP model there are those that correspond to fact and action *mutexes*, that are derived from the planning graph.

A *(Linear) Pseudo-boolean constraint* (PB-constraint) is an inequality of the form $\sum a_i x_i \geq b$, where x_i is a boolean variable, and a_i, b integers. A *Pseudo-boolean optimization* (PBO) problem on a set of variables X consists of a set of PB-constraints on X, together with a linear function on boolean variables from X. In PBO we seek a value assignment on the variables of X that maximizes the value of the boolean function, called objective function, and satisfies all the PB-constraints. The current implementation of PSP−H uses minisat+ (Eén and Sörensson 2006), to translate PB-constraints into CNF, and then invokes precosat (Biere 2009), that solves the SAT instance.

PSP−H is implemented as an extension of the PSP planning system (Sideris and Dimopoulos 2012). PSP follows a similar to the classic solve and expand approach, with the difference that it seeks to maximize the number of goals that can be achieved within a fixed planning horizon of, say, k steps. If this number is smaller than some user supplied value, the planning horizon is extended to $k + 1$ steps, and the procedure iterates. Moreover, the goals that are attained within the k steps are added to the problem as *intermediate goals*, that must be established in the $k + 1$ steps long plan that is generated in the next iteration. For a more detailed description of PSP the reader is referred to (Sideris and Dimopoulos 2012).

The PSP−H algorithm

PSP−H takes as input a (STRIPS) planning problem $(I, G, \mathcal{A})$, and a number of parameter values in the range [0..1]. The main technique implemented in PSP−H is the *incremental goal achievement*, that is embedded in its optimization phase. The core of the computation that is carried out in this phase is the solution of a series of PBO problems, where the maximization objective is $g_1(T)+\ldots+g_l(T)$, where $g_1, \ldots, g_l$ are the goal atoms and T the planning horizon. Obviously, the highest value that can be assumed by the objective function is l, which we denote by f_{max}.

PSP−H is also supplied by the user with an objective function value threshold $thres$. It starts with an initial planning horizon o that is determined by information that is extracted from the underlying planning graph as in the classic SATPLAN system. PSP−H solves the PBO with horizon o and obtains an optimal value f for the objective function. If $f < thres * f_{max}$, the planning horizon is set to $o + 1$, and the procedure repeats. If for some horizon o_b a solution is found with a value f_b for the objective function such that $f_b \geq thres * f_{max}$, a $plan$ is extracted from the solution. At this point, the state I' that results after the execution of the actions in $plan$ becomes the new initial state, and PSP−H repeats the process with a new planning problem $P' =< I', G, \mathcal{A} >$, where $P =< I, G, \mathcal{A} >$ and I are the original problem and initial state respectively. Moreover, the objective value threshold $thres$ is increased and set to $thres+restart\_rate$, where $restart\_rate$ is a user supplied value, called *restart rate*. As the maximization objective is defined on the fact variables that correspond to goals, the increase in the threshold translates into an increase in the number of goals that are achieved. Hence the term *incremental goal achievement*.

The optimization phase of PSP−H terminates when the value $thres$ exceeds another user supplied value $finphase$. Its output is a plan that is formed by concatenating the partial plans that are generated during its iterations. The execution of the action in this plan defines a state I_{sat}. Then PSP−H, similarly to PSP, enters the satisfaction phase, where a new planning problem with initial state I_{sat}, is solved by SMP.

The periodic replacement of the initial state of the planning problem described above, is called *restart strategy*, where the value of the $restart\_rate$ parameter determines the frequency of the restarts. With the restart strategy PSP−H attempts to limit the length of the planning horizon, and hence the size of the satisfiability instances that are solved by the system. As it is often the case with greedy search methods, a difficulty with incremental goal achievement is that it tends to generate locally optimal plans, i.e. plans that maximize the number of attained goals in the limited planning horizon, and ignores goals that are not achievable within that horizon. This bias may lead to poor quality solutions, as well as insolvability when actions interact in complex ways, as in the case of the presence of non-renewable resources.

To tackle this problem, PSP−H combines incremental goal achievement with a heuristic method that is intended to

account for the effort that is needed to achieve the remaining goals. This is accomplished by a problem model that consists of the usual layers of action and fact propositions that correspond to different times/points, but it is divided in two parts: the *constrained* and the *relaxed* part.

The constrained part is the standard PB model of the problem, i.e. a straightforward translation of the SMP clauses into linear inequalities. The relaxed part is also obtained from the SMP encoding, but some of the constraints of the model are omitted. In fact, as depicted in Figure 1, the relaxed part is further divided into three segments that differ in the mutual exclusion information they contain.

The first segment of the relaxed part is the *action relaxation*, which is the SMP model without action mutex clauses, and it is similar to the "$\exists$ encoding" of (Rintanen, Heljanko, and Niemelä 2006). The last segment is the *full relaxation* that contains neither action nor fact mutexes. This segment corresponds to the well-known delete-lists relaxation heuristic that forms the basis of many heuristic search planners. All goals are added as unit clauses in a final layer $q + 1$ (layer numbering is taken from figure 1).

Between the above two segments lies the *intermediate relaxation* segment. It is less constrained (more relaxed) than the action relaxation, but more constrained (less relaxed) than the full relaxation. It contains no action mutexes, and only a subset of the fact mutexes that is heuristically selected by PSP−H. This selection process is based on the analysis of the multi-valued variable representation of the problem that is carried out with the help of the preprocessor of LAMA (Richter and Westphal 2010). For some problem variables all corresponding mutexes are omitted, for others all mutexes are added, whereas for some variables x with domain values $x_1, \ldots, x_n$, instead of the constraint $\sum_{i=1}^{n} x_i <= 1$, PSP−H adds the weaker inequality $\sum_{i=1}^{n} x_i <= k$, for heuristically selected values of k. Due to space limitation, a more complete description of the intermediate relaxation falls outside the scope of this paper.

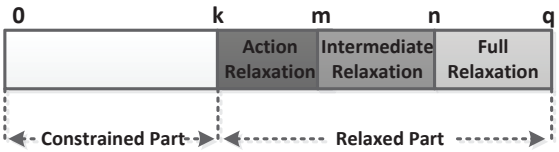

Figure 1: The different part of the encoding in PSP−H.

From the above discussion it must have become apparent that the problem model becomes less constrained as we move from left to right in figure 1. All the constraints of the different parts must be satisfied during the optimization phase of PSP−H, while the objective function that is maximized is $g_1(k) + \ldots + g_l(k)$, i.e. the goals refer to the last layer of the constraint part. The action variables of the first k layers (ie. the constrained part) that are assigned true in a solution generated by PSP−H in an iteration of the optimization phase, form a valid plan that achieves some of the goals at time k. The execution of the actions of this plan defines a valid state s_k at time k. As noted in the beginning of this section, this state s_k may become the initial state of the new problem that is solved in the next iteration of the optimization phase. The PSP−H encoding ensures that a state that satisfies all goals is reachable from s_k within $q + 1 - k$ *relaxed* steps. Therefore, PSP−H searches for a state s_k for which there is not only a valid plan from the current initial state, but there is also a relaxed, heuristically specified, plan from s_k to a final state. Hence the term *heuristic guided optimization*.

The computation of the length of the full relaxation segment takes into account a user supplied value for parameter h_init_perc, and is carried out as follows. PSP−H first removes all delete-lists from all problem actions, and then builds a planning graph until all goals are reachable. If the number of layers of this graph is len, the length of the relaxation part (corresponding to value $q - n$ in figure 1) is set to $len * h\_init\_perc$.

The length of the other segments of the relaxed part are computed in a similar way. First, PSP−H builds a planning graph of the original problem, and extends it until all goals are reachable. If the graph has len layers, the length of the action relation part is set to $len * r\_init\_perc1$ and the intermediate relaxation part to $len * r\_init\_perc2$, where again r_init_perc1 and r_init_perc2 are input parameter values.

In order to enhance the effects of heuristic guided optimization, PSP−H shortens the relaxed part periodically as the planner approaches a final state. The reduction always starts with the full relaxation segment, while its rate is determined by an input parameter h_red_rate. Roughly speaking, each time that the value of the objective function that is maximized in the optimization phase improves by h_red_rate, the full relaxation segment is reduced by one step. When its length reaches the value 1, the intermediate relaxation segment is reduced in a similar way. Action relaxation is never reduced.

Experimental evaluation

PSP−H is a prototype system that has been implemented to test the ideas that are presented in this work. It has been built on top of SATPLAN, from which it inherits the very slow and memory demanding planning graph construction phase. This imposes serious limitations on the size of the problem that can be solved by PSP−H. In fact, in many domains the current version of PSP−H spends more time in manipulating data structures than in constraint solving and optimization.

PSP−H has been evaluated on a number of domains from various planning competitions and compared with LAMA and Madagascar. All experiments were run on a server with 24 X5690 cores at 3.47GHz running under CentOS. Both LAMA and Madagascar were run with their default parameter values, and a CPU cutoff limit of 3600 seconds. In all experiments with PSP−H, the optimization phase terminated after 300 CPU seconds, in which case the planning horizon was extended by one step. PSP−H was also run with a CPU limit of 3600 seconds.

Table 1 summarizes the number of problems solved by each system. The first column lists the domains of the ex-

Domain	Num. of Probl.	LAMA	Madag	PSP-H
Pipesworld	25	25	22	25
Satellite	22	22	22	22
Storage	29	20	29	29
Openstacks	30	30	18	18
Elevators	30	30	30	30
Transport	19	19	19	18
Visitall	28	28	19	21
Barman	40	40	28	40
Total	223	214	187	202

Table 1: Number of problems solved by the planners within a 3600 seconds CPU limit. The second column lists the number of problems tried in each domain.

perimental evaluation, with the name with which they are known in the literature. The second column lists the number of problems included from each domain, as it was determined by PSP-H's memory consumption. Problems which exceed the memory capabilities of PSP-H are excluded from the study. The third and fourth column present the problems solved within the CPU cutoff limit by LAMA and Madagascar respectively. The last column refers to a characteristic run of PSP-H (the specific parameter values of this run are not of particular interest) that feature frequent restarts, thus making the satisfiability problems easier for the underlying boolean optimizer.

A comparison of PSP-H's column with that of LAMA, clearly shows that the heuristic planner outperforms PSP-H. In fact, the only domain where PSP-H (and Madagascar) solves more problems is Storage. On the other hand, PSP-H can solve more problems than Madagascar in Pipesworld, but its advantage shows more clearly in the Visitall and Barman domains, and this is not accidental. The plans in these domains are long, and it seems that the techniques employed by PSP-H address successfully this problem, at least in certain cases. We note that the longest plan that PSP-H was able to synthesize comes from Visitall and was 248 steps long.

Table 2 presents results about the quality of the plans that were generated by the systems for the largest 5 problems in each domain that were solved by all planners. The numbers in parentheses are the sums of the number of actions in all the 5 problems, whereas the numbers outside parentheses sum the parallel plan lengths, and hence they are not meaningful for LAMA which generates sequential plans. Clearly, PSP-H generates better quality plans in all domains except Satellite. On the other hand, in some domains with low or no action parallelism, such as Openstacks, Visitall and Barman, PSP-H generates plans that have less actions than those of LAMA. However, in domains with high parallelism, LAMA's plans contain considerably fewer actions.

Domain	LAMA	Madag	PSP-H
Pipesworld	(167)	193 (432)	152 (281)
Satellite	(366)	99 (457)	109 (527)
Storage	-	450 (1339)	194 (725)
Openstacks	(662)	448 (606)	396 (572)
Elevators	(839)	428 (2056)	382 (1629)
Transport	(341)	432 (1161)	220 (666)
Visitall	(448)	535 (535)	447 (447)
Barman	(814)	896 (1184)	561 (780)
Total	(3637)	3031 (6431)	2267 (4902)

Table 2: Sums of the solution length (number of parallel steps) and in parentheses the sums of the number of actions for the largest five problems in each domain. In the last line the total counts for all domains except Storage.

References

Biere, A. 2009. P{re,ic}oSAT@SC'09. In *SAT Competition 2009*, http://fmv.jku.at/precosat/.

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

Eén, N., and Sörensson, N. 2006. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation* 2(1-4):1–26.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *J. Artif. Int. Res.* 14(1):253–302.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013a. Red-black relaxed plan heuristics. In *AAAI*, 489–495.

Katz, M.; Hoffmann, J.; and Domshlak, C. 2013b. Who said we need to relax all variables? In *ICAPS*, 126–134.

Kautz, H. A., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic and stochastic search. In *AAAI*, 1194–1201.

Kautz, H. A.; Selman, B.; and Hoffmann, J. 2006. SATPLAN: Planning as satisfiability. In *Booklet of the 5th Planning Competition*.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res. (JAIR)* 39:127–177.

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

Rintanen, J. 2012. Planning as satisfiability: Heuristics. *Artif. Intell.* 193:45–86.

Sideris, A., and Dimopoulos, Y. 2010. Constraint propagation in propositional planning. In *ICAPS*, 153–160.

Sideris, A., and Dimopoulos, Y. 2012. Propositional planning as optimization. In *ECAI*, 732–737.

Vidal, V., and Geffner, H. 2006. Branching and pruning: An optimal temporal POCL planner based on constraint programming. *Artif. Intell.* 170(3):298–335.