Underestimation vs Overestimation in SAT-based Planning

Mauro Vallati, Lukáš Chrpa, and Andrew Crampton

School of Computing and Engineering
University of Huddersfield, United Kingdom
{m.vallati,l.chrpa,a.crampton}@hud.ac.uk

Abstract. Planning as satisfiability is one of the main approaches to finding parallel optimal solution plans for classical planning problems. Existing high performance SAT-based planners are able to exploit either forward or backward search strategy; starting from an underestimation or overestimation of the optimal plan length, they keep increasing or decreasing the estimated plan length and, for each fixed length, they either find a solution or prove the unsatisfiability of the corresponding SAT instance.
In this paper we will discuss advantages and disadvantages of the underestimating and overestimating techniques, and we will propose an effective online decision system for selecting the most appropriate technique for solving a given planning problem. Finally, we will experimentally show that the exploitation of such a decision system improves the performance of the well known SAT-based planner SatPlan.

1 Introduction

Planning as satisfiability is one of the main approaches to solving the classical planning problem in AI. Planning as satisfiability is commonly used for finding parallel optimal solution plans, and it gives the advantage of reusing the large corpus of efficient SAT algorithms and makes good use of the extensive advancement in SAT solvers. Approaches using this technique compile a classical planning problem into a sequence of SAT instances, with increasing plan length [10–12, 8, 9, 16]. The forward level expansion search keeps increasing the estimated plan length and for each fixed length finds a solution or proves the unsatisfiability of the SAT instance. While exploiting this approach, most of the search time is spent proving unsatisfiability of generated SAT instances, which could be expensive because the entire search space may need to be explored.

A different approach to SAT-based planning is exploited in MaxPlan [21]. This planner estimates an upper bound of the optimal plan length, by using the suboptimal domain-independent planner FF [7], and then exploits a backward search strategy until the first unsatisfiable SAT instance is reached.

Since both the approaches (the forward and the backward search) have advantages and disadvantages, which will be discussed in the paper, we designed a technique for combining them by selecting online the technique to exploit. The resulting system is called $\mathsf{SP}_{\mathsf{UO}}$ (SatPlan Under-Overestimating).

We focus our study on SatPlan [10–12], one of the most popular and efficient SAT-based optimal planning systems. Our experimental analysis will show that $\mathsf{SP}_{\mathsf{UO}}$ is able to efficiently select the most promising strategy, and that it is able to improve the performance of SatPlan.

The rest of the paper is organized as follows. We first provide some background and further information on SatPlan and SAT-based planning. Next, we describe the forward and backward approaches and we present in detail our experimental analysis and results, followed by concluding remarks and a discussion of some avenues for future work.

2 Background

A *classical planning problem* can be described as a tuple $\langle F, G, I, A \rangle$, where F is a set of facts (or state variables) that represent the state of the world, G is a set of facts representing the goal, I is a set of facts describing the initial state and A is a set of actions, that represents the different possibilities of changing the current state, described by their preconditions and effects.

A valid solution plan is a sequence $\langle A_1, \ldots, A_n \rangle$ of sets of actions, where the i-th set A_i represents the actions that can be executed concurrently at the i-th time step of the plan. An optimal (parallel) plan solving a classical planning problem is formed by the shortest sequence of action sets, and hence it can be executed using the minimum number n of time steps.

A well known and efficient optimal parallel planner for classical planning is SatPlan, which essentially computes the optimal plan by encoding the planning problem into a set of SAT problems that are solved by a generic SAT solver.

In particular, SatPlan first estimates an initial bound k on the length of the optimal plan by constructing the planning graph of the problem, as defined in GraphPlan [2]. A planning graph is a directed acyclic levelled graph representing, for each time step until the estimated horizon, the sets of facts that can be achieved, the set of actions that can be planned/executed, including special actions called "no-ops" used to propagate the facts forward in time across the time steps, and sets of mutually exclusive relations (called mutex relations) between facts and actions. Then, SatPlan encodes the planning problem with horizon equal to k into a propositional formula in Conjunctive Normal Form (CNF), and solves the corresponding SAT problem. Finally, SatPlan uses an incorporated SAT solver to solve this SAT problem. If the CNF is satisfiable, the solution to the SAT problem is translated into a plan, otherwise, SatPlan generates another CNF using an increased value for the planning horizon, and so on, until it generates the first satisfiable CNF. The unsolvability of the planning problem can be proved during the construction of the planning graph.

SatPlan has a modular architecture; it can work with different alternative SAT solvers and methods for encoding the planning problem into a CNF. In this work we consider, respectively, PrecoSAT [1] and the SAT-MAX-PLAN encoding [18]. PrecoSAT was the winner of the Application Track of the 2009 SAT Competition[1] and has been successfully exploited in several SAT-based approaches for planning, e.g., [18, 9, 19].

[1] http://www.satcompetition.org/

It uses a version of the DPLL algorithm [4] optimized by ignoring the "less important" generated clauses and by using on-the-fly self-subsuming resolution to reduce the number of newly generated clauses. The SAT-MAX-PLAN encoding is a very compact encoding that is directly derived from the planning graph structure. The variables used in the encoding represent time-stamped facts, actions and no-ops; the clauses are: unit clauses representing the initial and final state, clauses representing the link between actions to their preconditions and effects, and clauses expressing a compact representation of the mutex relations between actions and between facts in the planning graph. This encoding has been designed for both reducing the size of the SAT instance, by removing redundant clauses, and promoting unit propagation during the solving process.

3 Forward and Backward Approaches

As described in the previous section, the SatPlan planning system follows an incremental scheme. It starts to solve a planning problem by estimating a lower bound k of the length of the optimal plan, and moves forward until it finds the first satisfiable CNF. In this way SatPlan is able to find the optimal solution or to fail. This forward search approach has three main advantages: (i) it does not require any knowledge on the length of the optimal solution of the planning problem, (ii) the demonstration of optimality of the solution found is given by the planning process; a valid solution shorter than the one found does not exist, and (iii) it avoids the generation of a big CNF corresponding to a longer, w.r.t. the number of time steps encoded, planning graph. The last consideration is especially true while using old encoding strategies (e.g. [11]). On the other hand, new, compact encodings based on the planning graph structure have been proposed (e.g., SAT-MAX-PLAN encoding or the one proposed in [16]), so the size of the CNF is no longer a very constraining element of the SAT-based planning approach. Moreover, there is another fact that should be considered:

- It has been proven that on random SAT instances, unsatisfiable instances tend to be harder, w.r.t. the CPU-time needed, than satisfiable ones [14]. One reason for this behavior is that on satisfiable instances the solver can stop as soon as it encounters a satisfying assignment, whereas for unsatisfiable instances it must prove that no satisfying assignment exists anywhere in the search tree. No studies about the hardness of instances, w.r.t. satisfiability, have been done on structured instances, where propagation and clause learning techniques may have a lot more leverage.

Given this consideration, it seems reasonable that instead of starting from a lower bound of the optimal plan length, it could also be useful to overestimate the optimal plan length and start from a satisfiable SAT instance. This approach, that was first proposed in MaxPlan, has a number of distinct characteristics that make it very interesting. (i) The planning system, even if it does not find the optimal solution, can output suboptimal good quality solution(s), (ii) it is possible to avoid many hard, unsatisfiable instances, that become harder and harder as the system gets closer to the phase transition on solvability, (iii) it is possible to find a preliminary suboptimal solution plan and apply optimisation techniques, as done in MaxPlan, and (iv) it is possible to *jump*; while overestimating it could happen that the SAT-solver finds a satisfying variable assignment that corresponds to a valid solution plan that is shorter than the actual considered plan length. This means that some time steps are composed only by no-ops or unnecessary (w.r.t. the goal facts) actions. In this case it is possible to remove these actions, generate a shorter plan and jump closer to the optimal length.

In order to understand the actual usefulness of the described approaches, we have modified SatPlan. It is now able to exploit both the underestimation and the overestimation approaches. We decided to use SatPlan for two main reasons: it is well known and it has a modular architecture. The modular architecture allows us to suppose that the achieved results could be easily replicated with different combinations of SAT-solver and encodings. Moreover, since we are comparing search strategies and not planners, using the same planner should lead to the fairest possible comparison.

3.1 Related Works

The forward strategy was used in the original GraphPlan algorithm [2], and was then exploited also in SatPlan for finding makespan optimal plans.

The backward strategy, that still preserve the optimality of the solution found, was firstly presented in MaxPlan.

Alternatives to these strategies were investigated in [17]. In this work two algorithms were proposed: Algorithm A runs the first n SAT problems, generated with increasing time horizons, in parallel, each at equal strength. n is a parameter that can be tuned. Algorithm B runs all the SAT problems simultaneously, with the i^{th} problem receiving a fraction of the CPU time proportional to γ^i, where $\gamma \in (0, 1)$ is a parameter.

In [20], Streeter and Smith proposed an approach that exploits binary search for optimising the resolution process. The introduced approaches, Algorithms A and B, and the binary search, were shown to yield great performance improvements, when compared to the original forward strategy.

The aim of these works is to avoid hard SAT problems, i.e. CNFs that require a huge amount of CPU time for deciding about their solvability, for finding a satisficing solution as soon as possible. The solution found by their approaches are not proven to be optimal.

In this paper we are investigating the possibility of improving the performance of SAT-based planners from a different perspective. We are interested in preserving the demonstration of the optimality of solutions found, and we are studying the predictability of the slope of hardness of SAT problems from upper and lower bounds, w.r.t. the first satisfiable CNF of the planning problem.

4 Decision System

We have found that neither forward nor backward approach is always useful in speeding up the resolution process of SatPlan. Intuitively, the forward search is faster in very easy problems, where the CPU time needed for generating bigger CNFs becomes the bottleneck, or on problems that allow a lot of actions, where each new step implies a significant increase in the number of clauses and variables of the SAT instance. The

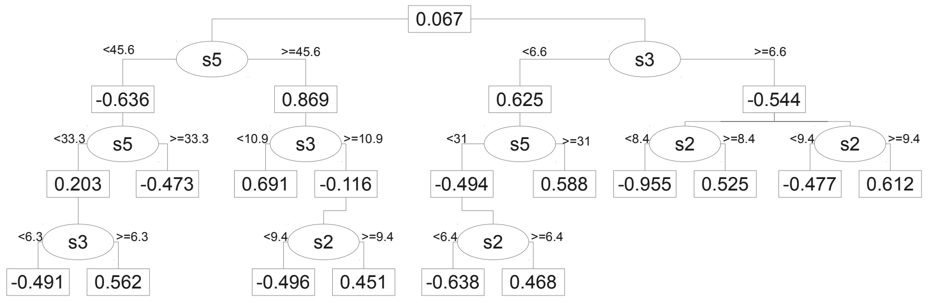

Fig. 1. The domain-independent online decision system represented as an alternating decision tree. Decision nodes are represented as ellipses; sY indicates that the decision is taken on the ratio of the set of clauses Y. Prediction nodes are represented as rectangles. An instance which obtains a positive score is classified as being suitable for the underestimation approach, otherwise the overestimation one is exploited.

backward search is faster, intuitively, on problems that have many different valid solutions since satisfiable instances tend to be easy to solve.

For deciding when it is useful to exploit the overestimating approach, and when it is better to use the underestimating one, we designed a decision system, that is shown in Figure 1. Firstly, we tried to extract information on the hardness of structured SAT instances of a given planning problem by evaluating the ratio of clauses to variables, following [13]. However, this ratio is not suitable for taking decisions because its value does not change much between different planning problems (especially from the same domain) and it does not seem to be in a direct relation with the hardness of satisfiable SAT instances generated from a planning problem. Instead, we divided the clauses of the SAT instances into 7 sets, according to information derived from the planning problem that they are encoding: (1) initial and final state, (2) linking actions to their preconditions, (3 & 4) linking actions to their positive and negative effects, (5 & 6) facts are true (false) at time step T if they have been added (removed) before, and (7) mutually exclusive actions and facts. We considered the ratios between the total number of clauses and each set of clauses of the SAT instance. This was done for avoiding too small numbers, that are potentially hard for machine learning techniques and, moreover, are hard to understand for humans. We considered these ratios because we believe they can give some insight about the hardness of structured SAT instances.

First of all, we did not consider the ratios of set 1. The number of these clauses is very small (usually around 20–30 in the considered problems) w.r.t. the size of the CNF (hundreds of thousands), and it does not change significantly across the considered training problems. On the other hand, we noticed that the value of some of the considered ratios remains almost the same throughout the SAT instances generated from the same planning problem, namely ratios of sets 2, 3, 5 and 7. This means that they are related to the planning problems, and moreover this fact allows us to evaluate only a very small SAT instance generated from the given planning problem for taking a decision.

Table 1. Min – Max values of each considered set of clauses on training problems of the selected domains.

Sets	Domains			
	Blocksworld	**Ferry**	**Matching-Bw**	**Satellite**
2	6.4–7.1	6.6–8.1	4.5–5.0	8.7–10.5
3	5.3–7.1	9.0–11.2	5.2–5.8	10.3–13.2
5	21.4–36.2	37.8–48.6	37.2–49.6	49.2–59.3
7	1.8–2.5	1.5–1.7	2.4–2.9	1.4–1.5

In Table 1 the minimum and maximum ratios of the considered clausal sets per domain are shown. The set 7 seems to be highly related to the structure of the domain, since there is no intersection between minimum and maximum values across the domains. Intuitively, one could agree that the information related to the mutexes is related to the domain. From this point of view, this set of clauses is not informative for an instance-specific predictive model.

For automatically finding correlations between the ratios and the effectiveness of the overestimating approach, we exploited an existing machine learning technique: the alternating decision tree (ADTree) [15], which is included in the well known machine learning tool Weka [6], due to its good performance on the training set. The resulting decision system is an alternating decision tree that is composed of decision nodes and prediction nodes. Decision nodes specify a condition on an attribute of the instance that is classified. Prediction nodes contain a single number. A new instance is classified by following all paths for which all decision nodes are true and by summing any prediction nodes that are traversed.

As training examples for the decision system we selected ten planning problems from `BlocksWorld`, `Matching-bw`, `Sokoban` and `Ferry` domains. For each of the forty problems we generated a SAT instance and extracted the ratios. Each planning problem has been solved using both forward and backward approaches, and classified w.r.t. their usefulness. We selected these domains because in `BlocksWorld` and `Ferry` the overestimating approach is very useful, and in the remaining two domains the underestimating approach is often the best choice. It should be noted that the training set is small. This is because we believe that it is easy, especially for an automatic machine learning tool, to extract useful information for classifying the planning problems. Moreover, a small training data set requires a small amount of CPU time for being generated and is easy to evolve in case of future improvements of the decision system.

The generated tree is shown in Figure 1; it is domain-independent and can be used to decide online which approach is better to exploit for the given planning problem. A new instance is classified by the sum of the prediction nodes that are traversed; positive values lead to exploiting the underestimating approach, negative values lead to the overestimating approach. For being used online, the CPU-time needed for taking this decision is critical; given the fact that the ratios are approximately the same on all SAT instances generated from the same planning problem, it is enough to generate the smallest CNF. The decision process usually takes between a tenth and a hundredth of a second.

By analysing the tree, it is possible to have some insight about which sets of clauses are relevant for taking the decision. Intuitively, the sets of clauses that are not changing their value too much through a single planning problem should be considered by the predictive model. In fact, it is easy to note that clauses encoding the connection between actions and facts are relevant (sets 2, 3 and 5), while the sets of clauses encoding the initial and goal states and, surprisingly, clauses encoding the mutually exclusive actions and facts are not significant for the purposes of this classification. This seems counter-intuitive; we are, essentially, trying to predict the hardness of satisfiable instances generated from a planning problem. Intuitively we would imagine some relations between hardness and mutually exclusive actions, but in fact the machine learning technique used, and some others tested, always ignore the ratio of this set of clauses since it does not change much throughout the considered training examples. From a closer look at the ADTree, we can also derive that, most of the times, the bigger is the number of clauses from the considered sets (which corresponds to smaller ratio), the most likely is the decision to overestimate. This lets us suppose that in problems where actions have many effects and require many preconditions, the overestimate approach is useful for achieving better performances.

5 Experimental Evaluation

All experimental tests were conducted using an Intel Xeon(tm) 3 GHz machine, with 2 Gbytes of RAM. Unless otherwise specified, the CPU-time limit for each run was 30 minutes (1800 seconds), after which termination was forced.

Overall, the experiments consider eight well-known planning domains: BlocksWorld, Depots, Ferry, Gripper, Gold-miner, Matching-bw, Sokoban and Satellite. These domains were selected because random instance generators are available for them, and because SatPlan is able to solve not only trivial instances, i.e., that have makespan shorter than 10 timesteps; very short plans are not significant for the scope of our study. Four of them, namely BlocksWorld, Ferry, Matching-bw and Satellite, have been considered for the training of the predictive model. We included them because we are interested in evaluating the performance of the generated model on both training domains and, obviously, new ones. For each selected domain we generated thirty benchmark problems using the available generator. These problems are different from those used for training the predictive model.

The performance of each approach was evaluated using the *speed score*, the performance score function adopted in IPC-7 [3], which is a widely used evaluation criterion in the planning community.

The speed score of a planning system s is defined as the sum of the speed scores assigned to s over all the considered problems. The speed score of s for a planning problem P is defined as:

$$\mathrm{S}core(s,P) = \begin{cases} 0 & \textit{if } P \textit{ is unsolved} \\ \frac{1}{1+\log_{10}(\frac{T_P(s)}{T_P^*})} & \textit{otherwise} \end{cases}$$

Table 2. Number of problems considered per domain (column 2), mean plan length (column 3), IPC score (columns 4-6), mean CPU time (columns 7-9) and percentages of solved problems (columns 10-12) achieved by respectively SatPlan, SP$_\mathsf{O}$ and SP$_\mathsf{UO}$, starting from -10/+10 time steps w.r.t. the optimal plan length.

Domain	#	PL	IPC Score			Mean CPU Time			% solved		
			SP$_\mathsf{U}$	SP$_\mathsf{O}$	SP$_\mathsf{UO}$	SP$_\mathsf{U}$	SP$_\mathsf{O}$	SP$_\mathsf{UO}$	SP$_\mathsf{U}$	SP$_\mathsf{O}$	SP$_\mathsf{UO}$
BlocksWorld	30	32.4	25.2	28.5	**28.6**	110.2	**81.5**	81.8	100.0	100.0	100.0
Ferry	30	29.2	11.1	**17.0**	**17.0**	309.4	**144.5**	**144.5**	46.7	**56.7**	**56.7**
Matching-bw	30	29.3	**22.0**	14.4	**22.0**	**261.1**	636.2	**261.1**	**73.3**	70.0	**73.3**
Satellite	30	23.7	**21.2**	19.0	**21.2**	308.0	**272.3**	305.3	73.3	73.3	73.3
Depots	30	16.3	**28.0**	12.7	27.0	**152.1**	724.1	197.9	**93.3**	70.0	**93.3**
Gripper	30	11.0	7.3	**9.9**	**9.9**	28.9	**12.5**	**12.5**	33.3	33.3	33.3
Gold-miner	30	55.2	27.4	**28.3**	27.7	**191.3**	207.2	193.1	100.0	100.0	100.0
Sokoban	30	38.7	28.2	27.6	**28.4**	64.8	64.2	**62.3**	100.0	100.0	100.0
All	240	24.9	170.4	157.4	**181.8**	175.3	267.9	**162.6**	77.5	75.4	**78.8**

where T_P^* is the lowest measured CPU time to solve problem P and $T_P(s)$ denotes the CPU time required by s to solve problem P. Higher values of the speed score indicate better performance. All the CPU times under 0.5 seconds are considered as equal to 0.5 seconds.

In the rest of this experimental analysis, SP$_\mathsf{O}$ stands for SatPlan exploiting the overestimation approach and SP$_\mathsf{U}$ stands for SatPlan exploiting the underestimation approach. The complete proposed approach is called SP$_\mathsf{UO}$; it is composed by the modified version of SatPlan, which is able to exploit both over/under-estimation, and the decision system for selecting online whether to overestimate or underestimate. The mean CPU times are always evaluated only on instances solved by all the compared systems.

In Table 2 are shown the results, in terms of IPC score, mean CPU times and percentages of solved problems of a comparison between the original SatPlan underestimating approach, SP$_\mathsf{O}$ and SP$_\mathsf{UO}$ on benchmark instances of selected domains. The overestimating approach starts solving from 10 time steps over the optimal plan length, while the original SatPlan underestimating approach is starting from 10 time steps lower. If the optimal plan is shorter than 10 time steps, the system is starting from the lower bound estimated by SatPlan. We are comparing the CPU times needed for demonstrating the optimality of the solution, i.e. if the optimal plan length is k, it must demonstrate the unsatisfiability of the planning problem using $k-1$ time steps.

The upper half of Table 2 shows the results on domains that were considered for training the predictive model exploited by SP$_\mathsf{UO}$, while in the lower half shows results on new domains. As expected, SP$_\mathsf{UO}$ always achieved the best IPC score in the upper half. But it is worth noting that also on new domains, it is usually able to achieve very good results. It is the best approach on two domains and it is very close to best in the remaining domains.

It is noticeable that in Ferry, the overestimating approach is always the best choice, since the IPC score has exactly the same value of the number of solved problems. In Gripper, it is almost the best approach, since on only one problem the underestimating

Table 3. For each considered domain, the percentages of solved instances in which it is better to apply the underestimating (column 2) or overestimating (column 3) approach, and the percentages of instances in which $\mathsf{SP_{UO}}$ exploited the right (fastest) approach (column 4).

Domain	**Under**	**Over**	$\mathsf{SP_{UO}}$
BlocksWorld	33.3	66.7	76.7
Ferry	0.0	100.0	100.0
Matching-bw	100.0	0.0	100.0
Satellite	80.0	20.0	72.3
Depots	100.0	0.0	90.0
Gripper	10.0	90.0	90.0
Gold-miner	50.0	50.0	53.3
Sokoban	60.0	40.0	70.0

approach has achieved better results. On the other hand, in planning problems from the Matching-bw and Depots domains underestimating is always the best choice. On the remaining domains, combining the two techniques is the best choice; in all of them there is a significant percentage of planning problems in which either the under or overestimating approach is faster.

Although in the presented results we considered only an under/over-estimation of 10 time steps, we experimentally observed that when the best approach is overestimating the optimal plan length, an overestimation of 20 time steps allows the system to achieve better results than an underestimation by only 10 time steps. On the other hand, we also experimentally observed that in the case in which it is better to underestimate, even an underestimation of 20 time steps allows the planning system to achieve better results than through even a small overestimation. Intuitively, we believe that this behavior derives from the fact that the hardness (or the gradient of the hardness) of SAT instances on these problems is very different between satisfiable and unsatisfiable ones.

Looking at the column with mean plan length (PL) over considered problems of each selected domain in Table 2, we can derive that the best strategy to exploit is not really related to the number of steps of the optimal plan. The overestimating approach achieved good results on both problems with short optimal plans (Gripper) and long optimal plans (Gold-miner).

For a better understanding of the relation between the two approaches on each of the selected domains, in Table 3 are shown the percentages of problems in which it is better to apply the underestimating or overestimating approach, and the percentages of instances in which $\mathsf{SP_{UO}}$ selected the right approach. The results shown in this table confirm the ones presented in Table 2; in the Matching-bw and Depots domains, underestimating is always the best choice, and in Ferry, overestimating is always better. But it is interesting to note that also in the other domains there is one technique which is better, at least in terms of the percentage of problems in which it is faster; this is the case for Sokoban and Satellite domains, in which underestimating is better, and BlocksWorld and Gripper, in which the overestimating approach allows us to achieve better results. In Gold-miner both the approaches are useful for speeding up the planning process on exactly 50% of the problems. Finally, $\mathsf{SP_{UO}}$ is generally able to select the right (fastest)

Table 4. For each considered domain, the IPC score (columns 2-4) and the mean CPU time (columns 5-7) of the proposed $\mathsf{SP}_{\mathsf{UO}}$, an Oracle selecting the best approach on benchmark problems and a random selection.

Domain	IPC Score			Mean CPU Time		
	$\mathsf{SP}_{\mathsf{UO}}$	Oracle	Random	$\mathsf{SP}_{\mathsf{UO}}$	Oracle	Random
BlocksWorld	28.6	**30.0**	26.5	81.8	**79.7**	110.2
Ferry	17.0	17.0	12.8	144.5	144.5	219.6
Matching-bw	22.0	22.0	14.6	261.1	261.1	512.7
Satellite	21.2	**22.0**	19.2	305.4	**234.1**	314.8
Depots	27.0	**28.0**	19.3	197.9	**152.1**	301.4
Gripper	9.9	**10.0**	8.1	12.5	12.5	28.3
Gold-miner	27.7	**30.0**	27.6	193.1	**166.9**	210.6
Sokoban	28.4	**30.0**	27.6	62.3	**59.2**	66.4
All	181.8	**189.0**	155.7	161.4	**142.0**	220.7

approach for solving a given problem. Only in Gold-miner it shows a precision below 70%. This is probably due to the fact that Gold-miner problems have usually a very similar structure, and that both SP_{U} and SP_{O} approaches lead to performances that are not significantly different.

In order to evaluate the accuracy of the decision system developed, we compared $\mathsf{SP}_{\mathsf{UO}}$ with an Oracle specifying the best approach to exploit for every benchmark problem and a random approach, that randomly selects the approach to use for the given problem. The results of this comparison are shown in Table 4. The decision system is very accurate on domains used for training the predictive model; on two domains it achieves the same IPC score of an Oracle, and on the remaining it is always very close to it. Also on new domains the decision system is able to achieve results that are very close to the Oracle ones. These are very interesting results, considering that the decision system is instance-based, is very fast, and has been trained on a small training set. The random selection approach usually achieved a low IPC Score; only in domains in which forward and backward approaches have similar behavior on considered problems e.g., Blocksworld, Gold-miner and Sokoban, its performance is close to the $\mathsf{SP}_{\mathsf{UO}}$ ones.

In terms of mean CPU times $\mathsf{SP}_{\mathsf{UO}}$ is almost always very close to the Oracle, while the random selected approach usually required a considerable amount of CPU time for solving testing instances. On Satellite the very high mean CPU time of $\mathsf{SP}_{\mathsf{UO}}$ is mainly due to only two instances (out of 22 solved) in which the selected approach is very slow.

6 Conclusions and Future Work

In this paper we investigated the usefulness of combining two well known approaches to planning as satisfiability: overestimating and underestimating. For combining these approaches, we proposed a machine learning based technique for taking instance-based domain-independent online decisions about the best strategy to exploit for solving a new planning problem. The decision system is represented as an alternating decision tree which evaluates the ratio of three sets of clauses w.r.t. the whole set of clauses of a SAT instance generated from the given planning problem. We have experimentally observed that the ratios are fixed throughout all the SAT instances generated from the same problem, so it is sufficient to generate the smallest one, which corresponds to the encoding of the first level of the planning graph in which all the goals are true, for deciding.

An experimental study, which considered 8 different domains and 240 planning problems, indicates that: (i) the best approach to exploit can be different across planning problems of the same domain; (ii) in planning problems in which overestimating is useful, an inaccurate upper bound performs better than a more accurate lower bound; (iii) the decision system is accurate, even if compared with an Oracle, and helps to effectively combine the underestimating and overestimating approaches; (iv) the $\mathsf{SP}_{\mathsf{UO}}$ system is very efficient and has outperformed SatPlan. We also observed that the overestimating approach usually is not useful on SAT encoding techniques that do not consider no-ops (see, e.g., [8]). Such encodings make it impossible to exploit jumps while overestimating, and moreover, we experimentally observed that it makes satisfiable instances, longer than the optimal one, harder to solve if compared to different encodings.

A limit of the current approach is that we have considered as known the optimal plan length, however, there already exists works about how to generate an accurate prediction of the optimal plan length (e.g. [5]) which could be exploited. Moreover, for overestimating the optimal plan length, the strategy adopted in [21] could be very useful; using a suboptimal planner to find a suboptimal sequential plan and then parallelize it. Finally, another strategy for overestimating the length of the optimal parallel plan, could be based on extracting information about the level of the fixed point of the planning graph of the given problem.

We see several avenues for future work on $\mathsf{SP}_{\mathsf{UO}}$. Concerning the optimal plan length prediction, we are evaluating the possibility of merging the proposed decision system with a predictive model of the optimal plan length. In this way, after deciding the approach to use, it can make a “biased” prediction for starting the planning process from a lower or an upper bound. Moreover, we are interested in evaluating the possibility of applying some sort of parallel solver in the proposed backward search; this has been already done in the SAT-based classical forward search in the well known Mp planner [16], with very interesting results. Furthermore, we are planning to study strategies for reusing part of the knowledge learnt by solving a satisfiable SAT instance, for speeding up the overestimating approach. Finally, we are interested in running additional experiments about the impact of the proposed $\mathsf{SP}_{\mathsf{UO}}$ on SAT-based planning, and in investigating the relation between the frequency and the length of *jumps* and the performance of the backward search.

Acknowledgements. The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

1. Biere, A.: P{re,i}cosat@sc’09. In: SAT Competition 2009 (2009)
2. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90, 281–300 (1997)

3. Celorrio, S.J., Coles, A., Coles, A.: Learning track of the 7th International Planning Competition. http://www.plg.inf.uc3m.es/ipc2011-learning (2011)
4. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)
5. Gerevini, A., Saetti, A., Vallati, M.: Exploiting macro-actions and predicting plan length in planning as satisfiability. In: Proceedings of the XIIth International Conference of the Italian Association for Artificial Intelligence (AI*IA-11). pp. 189–200. Springer (2011)
6. Hall, M., Holmes, F.G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)
7. Hoffmann, J.: FF: The Fast-Forward Planning System. AI Magazine 22(3), 57–62 (2001)
8. Huang, R., Chen, Y., Zhang, W.: A novel transition based encoding scheme for planning as satisfiability. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI-10). pp. 89–94. AAAI (2010)
9. Huang, R., Chen, Y., Zhang, W.: SAS+ planning as satisfiability. Journal of Artificial Intelligence Research 43, 293–328 (2012)
10. Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92). pp. 359–363. John Wiley and Sons (1992)
11. Kautz, H., Selman, B.: Unifying sat-based and graph-based planning. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99). pp. 318–325. Morgan Kaufmann (1999)
12. Kautz, H., Selman, B., Hoffmann, J.: SatPlan: Planning as satisfiability. In: Abstract Booklet of the 5th International Planning Competition (2006)
13. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In: Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92). pp. 459–465. AAAI (1992)
14. Nudelman, E., Leyton-Brown, K., Hoos, H.H., Devkar, A., Shoham, Y.: Understanding random SAT: Beyond the clauses-to-variables ratio. In: Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP-04). pp. 438–452. Springer (2004)
15. Pfahringer, B., Holmes, G., Kirkby, R.: Optimizing the induction of alternating decision trees. In: Proceedings of the 5th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD-01). pp. 477–487. Springer (2001)
16. Rintanen, J.: Engineering efficient planners with SAT. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12). pp. 684–689. IOS Press (2012)
17. Rintanen, J.: Evaluation strategies for planning as satisfiability. In: Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04). pp. 682–687. IOS Press (2004)
18. Sideris, A., Dimopoulos, Y.: Constraint propagation in propositional planning. In: Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS-10). pp. 153–160. AAAI (2010)
19. Sideris, A., Dimopoulos, Y.: Propositional planning as optimization. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI-12). pp. 732–737. IOS Press (2012)
20. Streeter, M.J., Smith, S.F.: Using decision procedures efficiently for optimization. In: Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS-07). pp. 312–319. AAAI (2007)
21. Xing, Z., Chen, Y., Zhang, W.: MaxPlan: Optimal planning by decomposed satisfiability and backward reduction. In: Proceedings of the 5th International Planning Competition (IPC-5) (2006)