Enhancing SAT Based Planning with Landmark Knowledge

Jan Elffers Dyan Konijnenberg Erwin Walraven Matthijs T.J. Spaan

Delft University of Technology, The Netherlands

Abstract

Several approaches exist to solve Artificial Intelligence planning problems, but little attention has been given to the combination of using landmark knowledge and satisfiability (SAT). Landmark knowledge has been exploited successfully in the heuristics of classical planning. Recently it was also shown that landmark knowledge can improve the performance of SAT based planners, but it was unclear how and in which domains they were effective. We investigate the relationship between landmarks and plan generation performance in SAT. We discuss a recently proposed heuristic for planning using SAT and suggest improvements. We compare the effects of landmark knowledge in parallel and sequential planning, also looking at previous research. It turns out that landmark knowledge can be beneficial, but performance highly depends on the planning domain and the planning problem itself.

1 Introduction

In Artificial Intelligence (AI) planning, one tries to find an action sequence from a given initial state to a goal state. The planning task is usually described in the high level Planning Domain Definition Language (PDDL) format [5]. This format consists of a description of the planning domain and the planning problem. The domain defines the predicates and actions that can be performed and the problem defines the initial state and the goal state. Multiple problems can be related to one domain. Several methods have been used to find plans. Examples are state-space search based methods and logic-based methods such as satisfiability (SAT). SAT encodings are a very powerful tool to express a wide range of combinatorial problems. Often, these problems can be translated to a propositional formula and subsequently be solved using a general SAT solver. In 1992, Kautz and Selman proposed SAT based AI planning and in 1996 they showed that satisfiability algorithms are a competitive alternative to the classical plan search approaches [3, 4]. In this paper we focus on factors that affect the efficiency of a planning task solver: landmark knowledge for planning and possible improvements for SAT planning heuristics.

Control knowledge is additional information inferred from the problem specification, which can be used to improve the efficiency of a solver, or to find a better plan. Specifically this control knowledge can be integrated in the encoding of a planning task by means of additional clauses. These clauses can help the SAT solver to find a solution more efficiently, in terms of running time. Recently, Cai et al. [1] proposed the idea of integrating landmark knowledge in the encoding of a planning task. In order to verify their observed increase in performance, we implemented their method using MiniSAT [2], a general purpose open-source SAT solver. Rintanen proposed several heuristics that can be used to make SAT solvers more planning specific, such as his own SAT solver Madagascar [11, 12]. Our work includes components of his solver Madagascar, MiniSAT and uses the LAMA planner to find landmarks. LAMA is a heuristic search planner using landmarks and winner of the 2008 International Planning Competition, which showed that landmarks can be succesfully used in planning [8, 10]. The paper also contains possible improvements for the heuristics proposed by Rintanen [12].

Our paper is organized as follows. In section 2 we give a general introduction to classical AI planning. Solving planning tasks with satisfiability solvers is the topic of section 3. Section 4 explains the concept of landmarks and section 5 contains possible improvements for planning heuristics. Next, the results of our experiments can be found in section 6. The last section contains our conclusion.

2 Planning

In this section an introduction to AI planning is given. In short, a planning task consists of an initial state and a goal state, which should be reached. A plan transitions from one state to another by executing actions. An action has preconditions that should be fulfilled before the action can be applied and there are postconditions (also called *effects*) that are established after executing the action. There are several definitions of a planning task, we use the definition presented in [12].

Definition 1 (Planning task [12]) *A planning task P is defined by a tuple $\langle X, I, A, G\rangle$, where X is a set of* ***state variables****, I is the* ***initial state****, A is a set of* ***actions*** *and G is the* ***goal state****. A state $s : X \rightarrow \{0, 1\}$ is an assignment of truth values to the variables in X. The actions in A are defined by a pair (p, e), where p and e are sets of literals representing the* ***preconditions*** *and the* ***effects*** *of the action. This means that the precondition literals must hold in order to apply the action and after applying the action the effect literals are true. Formally, an action (p, e) can be executed in state s if $s \models p$. This gives a state s' for which $s' \models e$ holds. Variables not affected by the effects remain unchanged.*

Given a planning task, a valid solution consists of actions that can be executed to achieve the goal state from the initial state. In this paper there are two variants of plans that are important. Sequential plans are a sequence of actions with only one action per time step. A sequential plan containing actions $0, 1, \ldots, t$ is feasible if $A_t(\ldots A_1(A_0(s))\ldots) \models G$, where $A_i(s)$ denotes the execution of action i in state s. Parallel plans are a sequence of sets of actions where every time step contains one or more actions that are executed. The actions defined at a time step can be executed in arbitrary order. This means that the outcome of the execution of the plan does not depend on the ordering in which the actions in parallel steps are applied.

In this paper the effects of landmark knowledge on sequential and parallel plans is compared. In order to run experiments we use problems formulated in PDDL [5].

3 SAT based planning

The satisfiability problem (SAT) is the problem to find an assignment of truth values to variables such that a propositional formula can be satisfied (i.e., evaluates to TRUE). The idea of translating a planning task to SAT was proposed by Kautz et al. [3]. In this section the reduction is explained, based on the notational conventions introduced by Rintanen [12].

The basic components are a parameter $T \geq 0$ that represents the number of time steps involved (also called the horizon). Each state variable $x \in X$ is represented by a variable x^t for each time point $t \in \{0, \ldots, T\}$. It indicates whether variable x is true at time t or not and represents the value of the state variable during the execution of a plan. If $X = \{x_1, \ldots, x_n\}$, then the state at time t is defined by the values of $x_1^t, \ldots, x_n^t$. The actions $a \in A$ are represented by a variable a^t for the time points $t \in \{0, \ldots, T-1\}$. Setting this variable to TRUE indicates that the action a is executed at time t.

Given a planning task and a horizon value T, the planning task can be translated to a formula that is satisfiable if and only if there exists a plan with horizon less than or equal to T. To include the preconditions and effects of action $a = (p, e)$ at time $t \in \{0, \ldots, T-1\}$, (1) is used. The first formula ensures that the preconditions have been fulfilled when executing the action and the second formula implies the relationship between executing the action and its effects for the next time step. Note that this assumes the precondition is one conjunction of atoms; for an arbitrary propositional formula one must use the Tseitin transformation [14] to map this to a clause set and then replace each atom l by l^t in the resulting clause set.

$$a^t \to \bigwedge_{l \in p} l^t \qquad\qquad a^t \to \bigwedge_{l \in e} l^{t+1} \tag{1}$$

The next component of the encoding contains information about variables that change or do not change, shown in (2). Suppose that $a_{1,x}, \ldots, a_{n,x}$ are the actions having x as one of the effects. Similarly, suppose that $a_{1,\neg x}, \ldots, a_{n,\neg x}$ are actions that have $\neg x$ as an effect.

$$x^{t+1} \to (x^t \vee a^t_{1,x} \vee \ldots \vee a^t_{n,x}) \qquad\qquad \neg x^{t+1} \to (\neg x^t \vee a^t_{1,\neg x} \vee \ldots \vee a^t_{n,\neg x}) \tag{2}$$

The first formula shows that if x is true at time $t+1$, this is caused by one of the actions with x as an effect, or it was already true at time t. The same reasoning can be used to explain the second formula.

The last part of the encoding contains information about the initial state and goal state. For all variables $x \in X$ that are true in the initial state a unit clause x^0 is added. Variables $x \in X$ that are false in the initial state correspond to unit clauses $\neg x^0$. For all literals that are true in the goal state we can add a unit clause x^T. The formulas discussed can be easily converted to Conjunctive Normal Form.

Parallel planning requires a more sophisticated formalism, and for more details about parallel encodings we refer to [13]. Our planning solver generates formulas for increasing values of horizon T and tries to find a satisfying assignment of truth values to variables. Once a satisfying assignment has been found, the corresponding plan can be deduced from the values of the action variables. Since our solver uses parts of the solver Madagascar, we use the included encodings for sequential plans and A-step plans [13]. The latter uses Graphplan parallelism, which is also used by the solver from [1]. Therefore, we argue that the encoding we use for parallel plans is similar.

4 Landmark knowledge

In this section we introduce the concept of landmarks and their SAT encoding. Landmarks were first defined by Porteous et al. [7] as "*facts that must be true at some point in every valid solution plan*". Later, Richter [8] extended the definition by changing '*facts*' to the more inclusive '*propositional formula*' to include disjunctive and conjunctive landmarks as well. Now, landmarks consisting of a single atom are called *unit* [1] or *fact* [8] landmarks. Of the previously mentioned types of landmarks, the unit (or fact) landmarks are used in our solver. The following definition of a landmark for SAT based sequential planning has been adhered to in this research:

Definition 2 (Landmark [9, 8, 1]) *Let $P = \langle X, A, I, G\rangle$ be a planning task and $\pi = \langle a^0, a^1, ..., a^t\rangle$ a sequential plan of P. A fact p is **true at time** i in π iff $p \in a^{i-1}(..a^0(I)...)$. A fact p is **added at time** i in π iff p is true at time i in π but not at time $i-1$. Facts in I are added at time 0. A fact p is **first added at time** i in π iff p is true at time i in π but not at any time $j < i$. A fact p is a **landmark** of P iff in each valid plan for P it is true at some time.*

Note that the facts in the initial and goal state are landmarks by definition and thus they are trivial landmarks. Cai et al. [1] proved that the definition of landmarks on sequential plans can be generalized to parallel plans and introduced a propositional encoding of landmarks (and their orderings). The knowledge of landmarks alone might not be sufficient to increase a solver's performance, but the orderings between them provide valuable information that can be used to form a plan. We have used the proposed encoding of landmark orderings as clauses in our SAT solver, which is explained further on. For that purpose we first define the considered landmark orderings.

Definition 3 (Orderings between landmarks [9, 1]) *Let q and p be landmarks of a planning task. If in each plan where p is true at time i and q is true at some time $j < i$, there is a **natural ordering** between p and q, written as $q_{nat} \to p$. If in each plan where p is first added at time i and q is true at time $i-1$, there is a **greedy-necessary ordering** between p and q, written as $q_{gn} \to p$. If in each plan where p is added at time i and q is true at time $i-1$, there is a **necessary ordering** between p and q, written as $q_{nec} \to p$.*

Our solver uses the landmark extraction from the LAMA planner [10]. More details about how the LAMA planner obtains landmarks and their orderings can be found in [8].

Landmark clauses Given the landmarks associated with a planning task and their orderings, these can be added as control knowledge to the SAT encoding in the form of additional clauses. Cai et al. [1] proposed the idea to encode landmarks orderings into clauses, which has been integrated in our solver and is discussed here. LAMA outputs a landmark graph, showing the orderings between all children and parent landsmarks. From this graph our solver generates additional clauses. A clause $p_0 \vee \ldots \vee p_T$ is generated for every landmark fact p, where the horizon of the current plan has been denoted by T. This clause ensures that the fact must be true at some point in any sequential plan, since it is a landmark. If we have two landmarks q and p with a greedy necessary ordering $q_{gn} \rightarrow p$, then we add the clause $q_{i-1} \vee \neg p_i \vee \bigvee_{m=i-1}^{m=0} p_m$ for $i = 1, \ldots, T$, where T is the horizon value. The first and second part of the disjunction represent the fact that q must be true at time $i-1$ if p is true at time i. However, it must only be the case if p becomes true for the first time at time i, so the last part of the clause represents the possibility that p was already true before.

The LAMA landmark detection procedure also identifies landmarks that are not a single unit (e.g., a landmark that is a disjunction of facts). These are not encoded in our planner. Cai et al. only considered clauses for facts and greedy necessary orderings in their experiments, because they found them to give superior results compared to other configurations [1]. In our experiments in section 6 we will do the same.

5 Variable selection heuristics for planning as SAT

In addition to the work done on landmark knowledge, we also present an idea for improving the variable selection heuristic of Madagascar. Madagascar is a solver for classical planning consisting of a procedure mapping PDDL instances to SAT instances, a built-in SAT solver for solving these, and a procedure scheduling the solving processes for various horizons. The built-in SAT solver is based on the Conflict Driven Clause Learning (CDCL) architecture [12]. One of the subroutines within this architecture is the variable selection heuristic: for a given state, which unassigned variable should be assigned (and which polarity should be chosen)? In this section, we first explain the scoring algorithm maintaining variable activities which forms the basis of the variable selection heuristic. Then, we consider the planning specific heuristic from [12] which Madagascar follows, and discuss possible improvements.

The scoring algorithm used by Madagascar is the Variable State Independent Decaying Sum (VSIDS) algorithm, which was first developed in the Chaff SAT solver [6]. The score of a literal at iteration t equals a geometric sum of the form $\sum_{t'} c[t'] \cdot 2^{-(t-t')}$, where t' ranges over the iterations during which a literal was *active*. A literal is active when it occurs in the implication graph during conflict analysis. Scores are not adjusted during unit propagation. These scores can be used directly for variable selection by choosing the maximum score literal; alternatively, they can be used as a building block in a more complicated procedure. Next, we discuss the planning specific heuristic of Madagascar, which uses these scores.

5.1 Madagascar's variable selection heuristic

There are two strategies possible in a backtracking search procedure to search for a plan: forward chaining (working from the initial state to the goal state; each path in the search tree is a plan *prefix*), or backward chaining (working from the goal state to the initial state; each path in the solving process is (the reverse of) a plan *suffix*). The "planning as SAT" approach seems to be neither forward nor backward chaining, because a satisfiability solver can assign variables in any order, so that the sequence of time points for which actions are assigned is not necessarily contiguous. It is not clear whether the possibility to form multiple subplans of actions "makes sense", that is, whether it leads to better search procedures compared to extending the plan from the front or the back. We verified experimentally that in the Madagascar solver, the solver sometimes is in a state where multiple subplans are formed.

Madagascar's variable selection heuristic resembles the method of backward chaining. The pseudocode for this procedure is given in [12] on page 9. Intuitively, the heuristic attempts to find *support* for (sub)goals recursively. The search starts at all goal literals l at time T. To find support for a literal at time t, l^t, one scans backwards in time, starting from time t, all $t' \leq t$ at which the state variable corresponding to l is assigned. There are two cases in which the scan terminates and the search for support continues recursively. If an action a is assigned at time t' with $l \in \mathrm{eff}(a)$, the scan terminates and support is recursively calculated for $\{l'^{t'} \mid l' \in \mathrm{prec}(a)\}$. The other case, if l is assigned FALSE at time t', one possible action a is chosen which has l as one of its effects. The (action, time) pair a^t is added to the set of candidate decision literals, and support is calculated for all literals in $\mathrm{prec}(a)$. When a termination criterion is met, for example when sufficient candidate literals are found, the search terminates and a candidate action is selected for branching.

5.2 Modification of the heuristic

In this section, we describe a modification to the above heuristic. Madagascar's heuristic can be seen as a search process in a graph: for each (subgoal) l^t, an action $\mathrm{support}(l^t) = a^{t'}$ is found, it is reported as a candidate decision literal if it is not yet assigned TRUE, and preconditions of a at time t' become new subgoals. The corresponding graph, with edges between state literals and actions, is formed by drawing directed edges from l^t to $\mathrm{support}(l^t) = a^{t'}$ and directed edges from $\mathrm{support}(l^t) = a^{t'}$ to all its preconditions at time t'.

The idea is to use the VSIDS scores to calculate "maximum activity paths" in this graph. The intuition is that activities of other nodes in the path should also be taken into account in the choice of the next action. For this purpose, we define a function $\mathrm{val}(p)$ which assigns values to paths p based on the activity scores $\mathrm{score}(l)$ for literals l. We considered functions parameterized by a real number parameter P, $0 \leq P \leq 1$:

$$\begin{aligned} \mathrm{val}([p_0]) &= \mathrm{score}(p_0) \\ \mathrm{val}([p_0, \ldots, p_{n-1}]) &= P \cdot \mathrm{score}(p_{n-1}) + (1-P) \cdot \mathrm{val}([p_0, \ldots, p_{n-2}]) \end{aligned}$$

These functions can be easily calculated recursively along paths, and they can be calculated recursively while the algorithm traverses the graph. The path weights can be used to determine which action to branch on. We implemented this modification, but unfortunately, we could not find benchmarks where setting $P \neq 1$ produced an improvement.

6 Experiments

In this section we present the results of our experiments with landmark clauses. Cai et al. [1] did an experimental evaluation for parallel plans. The purpose of our results is giving a complementary overview to place their results into perspective. In addition, we show that landmark knowledge can be useful for sequential planning as well and we identify differences between searching parallel and sequential plans.

There are three configurations of our solver that we use. MiniSAT is the standard configuration without control knowledge. MiniSAT-GN integrates clauses for greedy necessary orderings and MiniSAT-GN-F includes clauses for both facts and greedy necessary orderings. These configurations are similar to the configurations used in [1]. For each configuration we have a version for sequential and parallel plans. Our experiments have been run on a 3.2 GHz AMD Phenom II processor with a time limit of 15 minutes and a memory limit of 3.5 GB. We use the benchmark sets from the fifth IPC and the Fast Downward repository[1].

The SAT encoding used by [1] may be slightly different, but they also use an A-step encoding for parallel plans [13]. Cai et al. also count landmarks and orderings that do not correspond to propositional facts[2]. Our tables only include landmarks and orderings involving propositional facts.

6.1 Openstacks domains

Cai et al. observe that landmark clauses give a 50 percent increase in performance for the Openstacks domain [1]. We ran our solver on two sets of problems in this domain and observe different results. The

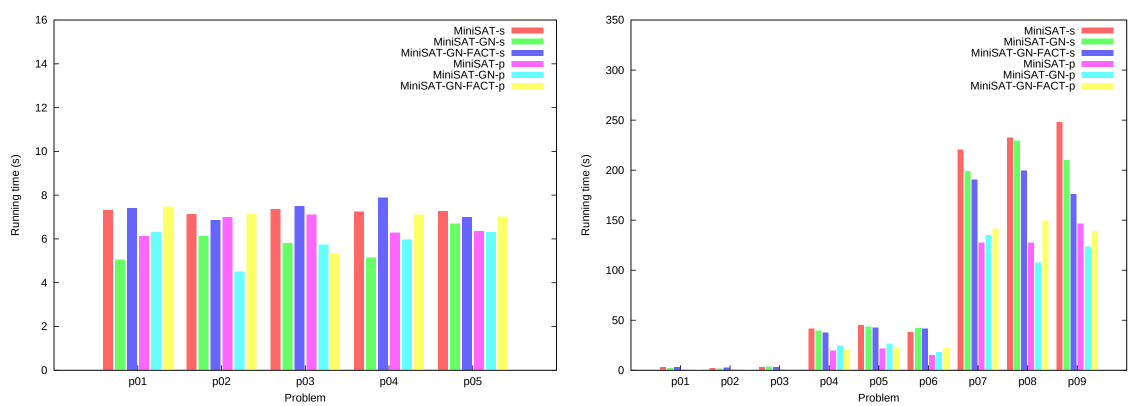
[1] Consult `http://zeus.ing.unibs.it/ipc-5` and `http://hg.fast-downward.org` for more info.

[2] This is not mentioned in their paper, but it has been verified through personal communication with the authors.

results are shown in Table 1 and Figure 1. LM and GN denote the number of landmarks and greedy necessary orderings, respectively. Running times are in seconds. We observe that adding landmark clauses can be beneficial for both parallel and sequential planning in this domain, but we do not observe such a large performance increase in Figure 1a. The last three problems in Figure 1b show again that landmark clauses can be beneficial, also for sequential plans (e.g., p07, p08 and p09), and it takes less time to find a parallel plan. Further investigation showed that MiniSAT made more restarts, conflicts and decisions when searching parallel plans (397, 651 and 546 percent, respectively, for p07 and M-GN).

Problem	**LM**	**GN**	**MS-s**	**MS-GN-s**	**MS-GN-F-s**	**MS-p**	**MS-GN-p**	**MS-GN-F-p**
Openst p01	31	45	7.32	5.06	7.40	6.14	6.30	7.48
Openst p02	31	45	7.14	6.12	6.86	7.00	4.52	7.14
Openst p03	31	45	7.36	5.82	7.50	7.12	5.74	5.34
Openst p04	31	45	7.24	5.16	7.88	6.28	5.96	7.10
Openst p05	31	45	7.26	6.70	7.00	6.36	6.30	7.02
Openst-sat08 p01	25	29	3.22	2.38	2.92	0.46	0.48	0.50
Openst-sat08 p02	25	29	2.18	1.84	2.46	0.48	0.52	0.52
Openst-sat08 p03	25	27	2.98	3.46	3.40	0.44	0.42	0.42
Openst-sat08 p04	50	60	41.52	39.50	37.48	19.80	24.52	20.06
Openst-sat08 p05	50	58	44.98	43.50	42.44	21.50	26.72	21.90
Openst-sat08 p06	50	60	38.36	42.10	41.44	15.24	18.14	22.32
Openst-sat08 p07	75	93	220.62	198.98	190.70	127.78	135.32	141.58
Openst-sat08 p08	75	89	232.74	229.70	199.48	127.42	107.42	149.82
Openst-sat08 p09	75	89	248.06	209.84	176.02	146.76	123.70	139.26

Table 1: Results for Openstacks problems.



(a) Openstacks

(b) Openstacks-sat08

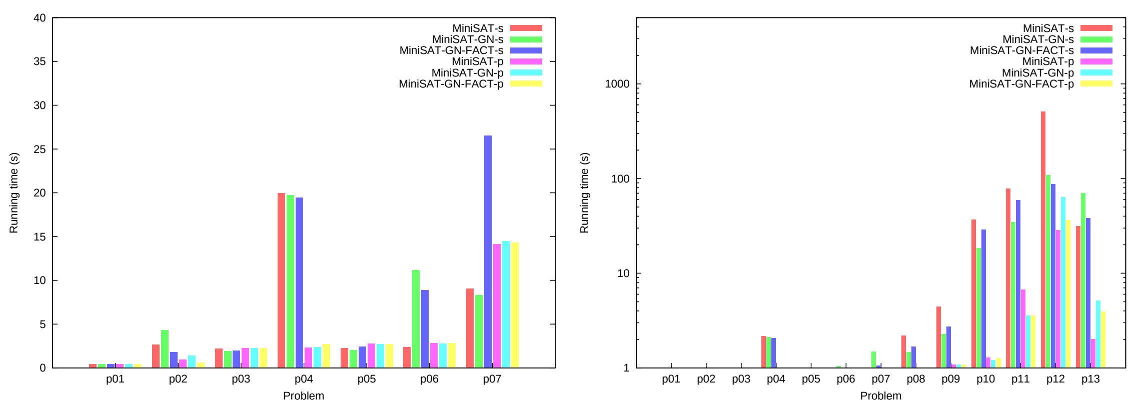
Figure 1: Results for Openstacks problems.

6.2 Pipesworld domains

The pipesworld domains come in two versions: tankage and notankage. The results in [1] show that landmark knowledge can be useful for specific problems in the pipesworld domains. Our results are shown in Table 2 and Figure 2 for some smaller problems. In essence, we observe the same thing, but there is no real trend that can be found. Consider, for example, p11 and p13 in Figure 2b. For sequential planning, GN orderings give an increase in performance for p11, while it is worse for p13. In Figure 2a we can see that fact clauses give a smaller running time for p06, but for p07 it performs badly. These examples show that the usage of landmark clauses highly depends on the nature of the problem itself, even though it is in the same domain.

Problem	**LM**	**GN**	**MS-s**	**MS-GN-s**	**MS-GN-F-s**	**MS-p**	**MS-GN-p**	**MS-GN-F-p**
Pipesworld-n p01	7	5	0.24	0.24	0.24	0.18	0.20	0.21
Pipesworld-n p02	12	8	0.94	0.86	0.88	0.26	0.24	0.24
Pipesworld-n p03	11	8	0.40	0.44	0.44	0.36	0.36	0.36
Pipesworld-n p04	15	10	2.16	2.12	2.06	0.38	0.40	0.38
Pipesworld-n p05	13	9	0.62	0.70	0.76	0.50	0.50	0.52
Pipesworld-n p06	18	12	0.66	1.04	0.68	0.52	0.56	0.54
Pipesworld-n p07	16	11	0.88	1.48	1.06	0.70	0.70	0.74
Pipesworld-n p08	22	15	2.18	1.46	1.68	0.72	0.76	0.80
Pipesworld-n p09	24	18	4.42	2.26	2.72	1.08	1.08	1.08
Pipesworld-n p10	32	24	36.74	18.34	28.72	1.28	1.20	1.26
Pipesworld-n p11	8	8	78.02	34.48	58.78	6.72	3.56	3.56
Pipesworld-n p12	12	13	505.94	108.68	87.10	28.42	63.28	36.06
Pipesworld-n p13	9	8	31.46	70.12	38.10	2.00	5.10	3.94
Pipesworld-t p01	13	12	0.44	0.44	0.44	0.42	0.42	0.42
Pipesworld-t p02	20	18	2.66	4.30	1.78	0.94	1.38	0.60
Pipesworld-t p03	17	14	2.20	1.92	1.94	2.24	2.24	2.24
Pipesworld-t p04	19	16	19.98	19.70	19.42	2.30	2.34	2.68
Pipesworld-t p05	19	16	2.26	2.02	2.40	2.74	2.68	2.72
Pipesworld-t p06	25	21	2.34	11.16	8.88	2.84	2.78	2.84
Pipesworld-t p07	17	11	9.04	8.30	26.52	14.10	14.44	14.32

Table 2: Results for Pipesworld problems.



(a) Pipesworld-tankage (b) Pipesworld-notankage (log scale)

Figure 2: Results for Pipesworld problems.

7 Conclusion

The effects of using landmark knowledge cannot be easily explained, but landmarks do show to increase performance on some types of problems and domains. At the moment it can be concluded that landmarks generally are useful on a small number of problems, but it seems that in some cases landmarks do not provide additional information. We were not able to reproduce the large increase in performance on the Openstacks domain from [1], but we observe that there is a significant improvement in some other cases. It confirms that their approach is promising, even with the Madagascar solver, but more research is required to make it working for a large range of problems. With that vision in mind, we identified research directions for improving planning specific heuristics in the SAT solver. As far as we know, there are currently no planning heuristics for SAT solvers exploiting landmark knowledge, so this can be considered as future work.

Acknowledgements

We would like to thank Jussi Rintanen and Dunbo Cai for sharing parts of their source code and their helpful correspondence.

References

[1] Dunbo Cai and Minghao Yin. On the utility of landmarks in SAT based planning. *Knowledge-Based Systems*, 36:146–154, 2012.

[2] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, pages 502–518, 2003.

[3] Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European conference on Artificial intelligence*, ECAI '92, pages 359–363. John Wiley & Sons, Inc., 1992.

[4] Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 2*, AAAI'96, pages 1194–1201, 1996.

[5] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition Language. Technical Report TR-98-003, Yale Center for Computational Vision and Control, 1998.

[6] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Annual ACM IEEE Design Automation Conference*, pages 530–535. ACM, 2001.

[7] Julie Porteous, Laura Sebastia, and Jorg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *Proceedings of the 6th European Conference on Planning (ECP 01)*, 2001.

[8] Silvia Richter. *Landmark-Based Heuristics and Search Control for Automated Planning*. PhD thesis, Griffith University, November 2010.

[9] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 975–982, 2008.

[10] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[11] Jussi Rintanen. Heuristics for planning with SAT and expressive action definitions. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling*, 2011.

[12] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[13] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12):1031–1080, September 2006.

[14] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, 1983.