Charles University in Prague

Faculty of Mathematics and Physics

ABSTRACT OF DOCTORAL THESIS

Tomáš Balyo

Modelling and Solving Problems Using SAT Techniques

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the doctoral thesis:	prof. RNDr. Roman Barták, Ph.D.
Study programme:	Computer Science
Specialization:	Theoretical Computer Science

Prague 2014

Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

AUTOREFERÁT DISERTAČNÍ PRÁCE

Tomáš Balyo

Modelling and Solving Problems Using SAT Techniques

Katedra teoretické informatiky a matematické logiky

Vedoucí doktorské disertační práce:	prof. RNDr. Roman Barták, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika (4I1)

Praha 2014

Výsledky doktorské disertační práce byly získány během interního studia na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze v letech 2010 – 2014.

Uchazeč:	RNDr. Tomáš Balyo
Vedoucí:	prof. RNDr. Roman Barták, Ph.D. Katedra teoretické informatiky a matematické logiky MFF UK Malostranské náměstí 25 118 00 Praha 1
Školící pracoviště:	Katedra teoretické informatiky a matematické logiky MFF UK Malostranské náměstí 25 118 00 Praha 1
Oponenti:	Univ.-Prof. Dr. Armin Biere Johannes Kepler University in Linz Institute for Formal Models and Verification Altenbergerstr. 69 4040 Linz Austria doc. Ing. Filip Železný, Ph.D. Katedra počítačů FEL ČVUT Karlovo náměstí 13 121 35 Praha 2

Autoreferát byl rozeslán dne 29. 8. 2014.

Obhajoba se koná dne 29. 9. 2014 v 10:00 hodin před komisí pro obhajoby disertačních prací oboru 4I1: Teoretická informatika na Matematicko-fyzikální fakultě v Praze, Malostranské náměstí 25, 118 00 Praha 1, v místnosti malá aula.

Disertační práce je k dispozici na studijním oddělení UK MFF, Ke Karlovu 3, 121 16 Praha 2.

Předseda RDSO:	prof. RNDr. Václav Koubek, DrSc. Katedra teoretické informatiky a matematické logiky MFF UK Malostranské náměstí 25 118 00 Praha 1

Contents

1. Introduction

One of the most studied problems in computer science, both theoretical and applied, is the Boolean satisfiability problem (SAT). SAT solvers have seen a lot of progress in the last two decades which allowed SAT solving to become a core component in many different applications. One of the first and most successful applications of SAT was automated planning [21].

Planning [17] is the problem of finding a sequence of actions – a plan, that transforms the world from some initial state to a goal state. In the thesis we consider only the simplest (most limited) definition of planning – often referred to as *classical* or *STRIPS* planning. The world is fully-observable, deterministic and static (only the agent, that we make the plan for, can change the world). The number of possible states of the world as well as the number of possible actions is finite, though possibly very large. We will also assume, that the actions are instantaneous (take a constant time) and therefore we only need to deal with their sequencing.

Other kinds of planning such as *temporal* and *probabilistic* planning [17], which remove these limitations, are also studied in the literature. Their advantage is that they model the real world more faithfully and thus are more applicable. On the other hand, solving these kinds of problems can be much harder and there are not many efficient planners capable of solving them.

The complexity of planning depends on the planning problem instance and the quality of the required plan. There are instances where a plan can be found in polynomial time while finding an optimal (best quality) plan is NP-hard [9]. However, in many cases (and in general) already finding a plan of arbitrary quality (but with polynomial length) is NP-complete [10]. Determining whether there is a solution (a plan) for a given planning task is in general PSPACE-complete [10].

Despite the complexity results many 'real world' planning tasks can be successfully solved by modern state-of-the-art planning systems. Satisficing planners such as FF [19], Fast Downward [18] or LPG [16] are very efficient on a wide range of problems. They are based on heuristic guided search of the state space of a given planning problem. Another approach, formerly very successfully used for optimal planning [22] and currently also for satisficing planning [27], is translating the planning task into a series of propositional satisfiability (SAT) formulas and then using a SAT solver.

In the thesis we deal with the problem of encoding a planning task to SAT in order to efficiently find its solution (a plan). We will also show how SAT and MaxSAT solvers can be used to improve the quality of an already obtained plan by removing useless (redundant) actions.

1.1 Contributions

The main contribution of the thesis is the definition of new encoding schemes for planning as SAT and their theoretical and experimental evaluation. These encodings are used for two purposes. One is finding plans and the other is improving plans found by other planning systems by removing redundant actions from them.

For the first purpose we introduce two new encoding schemes. One is called the Reinforced encoding and it slightly improves upon the existing so called $\forall$-step semantics based encodings. The other is a new innovative encoding called the Relaxed Relaxed $\exists$-step encoding which works well on planning problems that were previously very difficult to solve using SAT based techniques. We also define a simple rule, that given a planning problem instance can predict which encoding scheme is more suitable for solving the instance. Parts of these results are already published in a conference paper [2].

For the second purpose we introduce a propositional encoding for the problem of plan redundancy. We use this encoding to generate SAT and MaxSAT formulas which allows us to efficiently solve NP-hard plan optimization problems, which were previously only addressed by using heuristic algorithms. We also introduce our own heuristic algorithm which improves upon the existing ones. The results are already published [5] or accepted for publication [6].

2. Preliminaries

In this Chapter we give the basic definitions and properties related to satisfiability, maximum satisfiability and planning.

2.1 SAT and MaxSAT

A *Boolean variable* is a variable with two possible values *True* and *False.* A *literal* of a Boolean variable x is either x or $\neg x$, i.e., *positive* or *negative literal.* A *clause* is a disjunction (OR) of literals. A *conjunctive normal form (*CNF*) formula* is a conjunction (AND) of clauses. A *truth assignment* ϕ of a formula F assigns a *truth value* to its variables. The assignment ϕ satisfies

- a positive literal if it assigns the value True to its variable,
- a negative literal if it assigns the value False to its variable,
- a clause if it satisfies at least one of its literals,
- a CNF formula if it satisfies each one of its clauses.

If ϕ satisfies a CNF formula F, then ϕ is called a *satisfying assignment* of F. A formula F is said to be *satisfiable* if there is a truth assignment ϕ that satisfies F, i.e. ϕ is a satisfying assignment of F. Otherwise, the formula F is *unsatisfiable.* The *problem of satisfiability (SAT)* is to determine whether a given formula F is satisfiable or unsatisfiable.

A *SAT solver* is a procedure that solves the SAT problem. For satisfiable formulas we also expect a SAT solver to return a satisfying assignment. An example of a satisfiable CNF formula with its satisfying assignment follows.

Example 1. *$F = (x_1 \vee x_2 \vee \neg x_4) \wedge (x_3 \vee \neg x_1) \wedge (\neg x_1 \vee \neg x_2)$ is CNF formula with 3 clauses $\{(x_1 \vee x_2 \vee \neg x_4),\ (x_3 \vee \neg x_1),\ (\neg x_1 \vee \neg x_2)\}$ and 6 literals $\{x_1, \neg x_1, x_2, \neg x_2, x_3, \neg x_4\}$ on 4 variables $\{x_1, x_2, x_3, x_4\}$. F is satisfiable with $\phi = \{x_1 \rightarrow False, x_2 \rightarrow True, x_3 \rightarrow True, x_4 \rightarrow True\}$ being a satisfying truth assignment of F.*

The problem of *Maximum Satisfiability (MaxSAT)* is the problem of finding a truth assignment of a given CNF formula that satisfies the maximum number of its clauses.

A *partial maximum satisfiability (PMaxSAT) formula* is a CNF formula consisting of two kinds of clauses called *hard* and *soft* clauses. The *partial maximum satisfiability problem* (PMaxSAT) is to find a truth assignment for a given PMaxSAT formula that satisfies all the hard clauses and as many soft clauses as possible.

There are two special cases, one is that all the clauses are hard, the other is that all the clauses are soft. In the first case PMaxSAT is equivalent to SAT, in the second to MaxSAT.

In the situation, that not all the soft clauses are equally important, we can assign weights to them and we obtain the weighted partial MaxSAT formula. The *Weighted Partial MaxSAT (WPMaxSAT)* problem is to find a truth assignment for a given weighted partial MaxSAT formula that satisfies all its hard clauses and maximizes the sum of the weights of satisfied soft clauses.

2.2 Planning

In the introduction we briefly described what planning is, in this section we give the formal definitions. We will use the SAS+ formalism [1] based on multivalued state variables instead of the classical STRIPS formalism [14] based on propositional logic.

Definition 1 (Planning Task)**.** *A planning task* Π *in the SAS+ formalism is defined as a tuple* $\Pi = \langle X, O, s_I, s_G \rangle$ *where*

- $X = \{x_1, \dots, x_n\}$ *is a set of multivalued state variables with finite domains* $\mathrm{dom}(x_i) \subset \mathbb{N}$.

- *O is a set of actions (or operators). An action* $a \in O$ *is a tuple* $(\mathrm{pre}(a), \mathrm{eff}(a))$ *where* $\mathrm{pre}(a)$ *is the set of preconditions of a and* $\mathrm{eff}(a)$ *is the set of effects of a. Both preconditions and effects are of the form* $x_i = v$ *where* $v \in \mathrm{dom}(x_i)$*. The actions may have a non-negative integer cost assigned to them. We will denote by* $C(a)$ *the cost of an action a.*

- *A state is a set of assignments to the state variables. Each state variable has exactly one value assigned from its respective domain. We denote by S the set of all states.* $s_I \in S$ *is the initial state.* s_G *is a partial assignment of the state variables (not all variables have assigned values) and a state* $s \in S$ *is a goal state if* $s_G \subseteq s$.

An action can be applied to a state if its preconditions are satisfied. For example if $x_i = v$ is in the preconditions of an action, then the action can be applied only to states where the state variable x_i is equal to v. Similarly, the effects of an applied action change the world state. If $x_i = v$ is an effect of an action, then after the application of this action the value of x_i will become v. A state obtained by applying an action a to a state s is denoted by $\mathrm{apply}(a, s)$. If $A = [a_1, \dots, a_k]$ is a sequence of actions, then $\mathrm{apply}(A, s) = \mathrm{apply}(a_k, \mathrm{apply}(a_{k-1} \dots \mathrm{apply}(a_2, \mathrm{apply}(a_1, s)) \dots))$.

A *sequential plan* P of *length* k for a planning task $\Pi = \langle X, O, s_I, s_G \rangle$ is a sequence of actions $P = [a_1, \dots, a_k]; a_i \in O$ such that $s_G \subseteq \mathrm{apply}(P, s_I)$.

We will denote by $|P| = k$ the length of a plan P and by $P[i]$ we will mean the i-th action of P, i.e., $P[i] = a_i$. If P contains actions with costs, then we define the *cost of a plan*, $C(P)$, to be the sum of the costs of the actions in it.

A plan P for a planning task Π is called an *optimal plan* if there is no other plan P' for Π such that $|P'| < |P|$. Similarly, a plan P is called *cost optimal* if there is no other plan P' such that $C(P') < C(P)$.

Example 2. *In this example we will model a simple package delivery scenario. We have a truck that needs to deliver two packages to the location C from the locations A and B. In the beginning the truck is located in A. The locations A, B, and C are connected by roads.*

We will model the planning task using the following variables:

- *Truck location T, $dom(T) = \{A, B, C\}$*
- *Package locations P_1 and P_2, $dom(P_1) = dom(P_2) = \{A, B, C, L\}$ ($P_i = L$ represents that the package i is inside the truck).*

Now, having defined the variables $X = \{T, P_1, P_2\}$ and their respective domains, we can define the initial state s_I and the goal conditions s_G.

$$s_I = \{T = A, P_1 = A, P_2 = B\} \qquad s_G = \{P_1 = C, P_2 = C\}$$

Finally, we need to define the set of actions with their preconditions and effects. The action templates are described in the following table.

Action	*Preconditions*	*Effects*	*Description*
$move(l1, l2)$	$T = l1$	$T = l2$	*move the truck from location $l1$ to $l2$*
$loadP_1(l)$	$T = l, P_1 = l$	$P_1 = L$	*load P_1 on the truck at location l*
$loadP_2(l)$	$T = l, P_2 = l$	$P_2 = L$	*load P_2 on the truck at location l*
$unloadP_1(l)$	$T = l, P_1 = L$	$P_1 = l$	*unload P_1 from the truck at location l*
$unloadP_2(l)$	$T = l, P_2 = L$	$P_2 = l$	*unload P_2 from the truck at location l*

To get the actual actions we need to substitute $l, l1, l2$ with A, B and C.

One of the possible solutions is the following plan $P = [loadP_1(A), move(A, C), unloadP_1(C), move(C, B), loadP_2(B), move(B, C), unloadP_2(C), move(C, A)]$. The plan P is valid, but it is not an optimal plan. It contains 8 actions while the following (optimal) plan has only 6 actions: $P^* = [loadP_1(A), move(A, B), loadP_2(B), move(B, C), unloadP_1(C), unloadP_2(C)]$.

Suboptimal plans often contain actions that can be removed without affecting their validity. Such actions are called redundant. A plan reduction P' is a subsequence of a plan P with some redundant actions removed (first defined in [15, 24]). A plan which is not redundant is called a *perfectly justified plan.*

Example 3. *Using the package delivery planning problem of Example 2 we can observe, that the last action of P ($move(C, A)$) is redundant. By removing this action from the plan we get a perfectly justified plan P' which is a plan reduction of P. Nevertheless, P' is not optimal, since it contains 7 actions, while the optimal plan P^* from Example 2 contains only 6 actions. Note, that P^* is not a plan reduction of P, which means we could not get to P^* from P by just removing actions.*

The decision version of this problem (given a plan P and a number L, is there a plan reduction of length/cost at most L?) is NP-complete [15, 24]. From this it also follows, that just checking if a given plan is perfectly justified is NP-complete, which is equivalent to the question whether at least one action can be removed.

3. Finding Plans using SAT

The basic idea of solving planning as SAT is the following [22]. We construct (by encoding the planning task) a series of SAT formulas $F_1, F_2, \ldots$ such that F_i is satisfiable if there is a plan of length $\leq i$. Then we solve them one by one starting from F_1 until we reach the first satisfiable formula F_k. From the satisfying assignment of F_k we can extract a plan of length k.

The method was first introduced by Kautz and Selman [22] and is still very popular and competitive. This is partly due to the power of SAT solvers, which are getting more efficient year by year. Since then many new improvements have been made to the method, such as new compact and efficient encodings [20, 27, 28], better ways of scheduling the SAT solvers [27] or modifying the SAT solver's heuristics to be more suitable for solving planning problems [25]. Clever ways of solver scheduling [27] can also significantly improve the performance of the planning algorithm at the cost of possibly longer plans. Nevertheless, we will use the basic one-by-one scheduling since we are interested only in comparing the properties of encodings, i.e., the construction of the formulas F_i.

In the thesis we examine the following four encodings.

- *Direct Encoding.* The simplest and most straightforward way of encoding a planning task into SAT is following the definition of the planning problem and translating it into propositional logic. This encoding was the first one used in SAT based planners [21, 9]. Originally it was described in the context of the propositional planning formalism (STRIPS [14]). In the thesis we adapt it for the multivalued SAS+ formalism [1].

- *SASE Encoding.* The SASE Encoding was historically the first SAT encoding of a planning task based on the SAS+ formalism [20]. It is based on reasoning about the transitions of state variables rather than their values. The encoding we present in the thesis is slightly modified compared to the original SASE paper [20]. In particular, we use a different encoding of the initial state and goal conditions as well as the interference of transitions is defined in a more strict manner (see our recent paper [4] for more information about transition interference).

- *Reinforced Encoding.* This new encoding introduced in the thesis is a combination of the Direct and SASE encodings. The encoding shares all the variables and many of the clauses used in the Direct and SASE encodings. The name of the encoding comes from the idea of reinforcing one encoding with the strengths of the other. In other words, we are 'reinforcing' the Direct Encoding by using transitions, or alternatively, we are 'reinforcing' the SASE Encoding by using assignments.

- *$R^2\exists$-Step Encoding.* This encoding is very different from the previous three encodings since it uses a new parallel planning semantics, which means that fewer SAT calls are required to solve planning tasks. It was introduced in our recent paper [2]. This encoding requires a special preprocessing step – action ranking. Intuitively, the ranking should be such, that the actions are ranked according to their order in a valid plan for the given planning task. The problem is, of course, that we do not know the plan in advance. Therefore, we try to guess the order in which the given actions could appear in a plan using heuristics. In the thesis we show that the quality of the ranking dramatically affects the performance of the planning procedure and propose several ranking heuristics.

For each of these encodings we give exact definitions of the variables and clauses contained in the encoded formulas. We prove the correctness of the encodings and compute upper bounds on the number of Boolean variables and clauses in the constructed formulas.

While doing experiments with our new encodings we noticed, that on some problems the $R^2\exists$-Step encoding works very well and significantly outperforms the other encodings. But there are also problems, where the situation is reversed, i.e., the $R^2\exists$-Step encoding performs very poorly.

Our goal is, of course, to have an encoding that works well for all the problems, or at least for as many as possible. This can be achieved by combining two or more different encodings into one to form a selective encoding.

A *selective encoding* consists of a set of encodings E and a *selection rule*. The task of the selection rule is to select an encoding from E that should be used to solve a given planning task based on the planning task itself and other available information. A good selection rule should be simple, so it be can evaluated quickly and clever, so it can select the proper encoding for a planning task.

For our experiments we constructed a simple selective encoding which consists of the Reinforced and the $R^2\exists$-Step encoding.

3.1 Experiments

To compare the encodings to each other and to other state-of-the-art encodings we did experiments using all the benchmark problems from the optimization track of the 2011 International Planning Competition (IPC) [12].

The experiments were run on a computer with Intel i7 920 CPU @ 2.67 GHz processor and 6 GB of memory. Our five encoding procedures (four encodings and the selective encoding) were implemented in Java. To obtain the state-of-the-art $\forall$-Step formulas and $\exists$-Step formulas we used Rintanen's planner Madagascar[26] (version 0.99999 21/11/2013 11:54:15 amd64 1-core). For each encoding we used the same SAT solver – Lingeling[8] (version ats).

In Figure 3.1 we provide a so called cactus plot, that visually compares the five strongest encodings. The plot represents how many problems can be solved withing a given time limit. It is interesting, that the lines for the $R^2\exists$-Step and the 'Rintanen $\forall$' encoding cross each other several times, which means, that for certain time limits $R^2\exists$-Step would solve more instances than the 'Rintanen $\forall$'

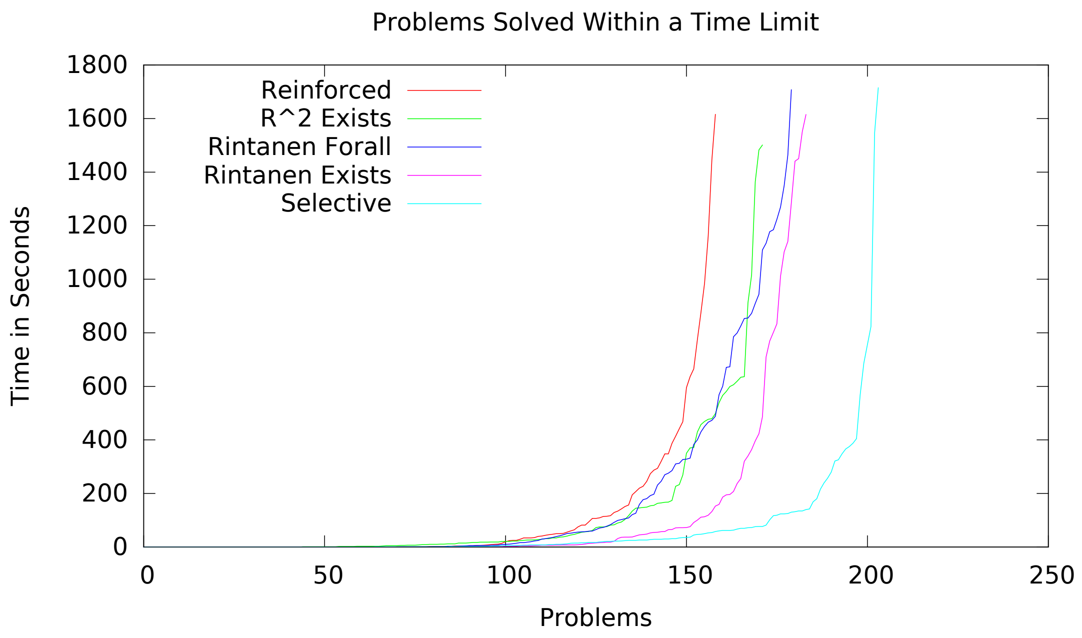

Figure 3.1: This plot represents the number of problems that a SAT based planner using the given encoding can solve under a given time limit. It is obtained by sorting the SAT solving times for the solved problems for each encoding.

encoding. The cactus plot shows, that the Selective method significantly outperforms all the other methods, furthermore, this holds for any reasonable time limit.

More detailed experimental results can be found in the thesis. Overall, the experiments revealed, that our new $R^2\exists$-Step encoding is very efficient for some domains, that other encodings cannot handle. On the other hand, its performance highly depends on the ranking of actions and it does not perform well on domains with a high number of transitions per variable. Fortunately, we could design a simple selective approach which can decide when the $R^2\exists$-Step encoding should be used and by trying several action rankings the problem of proper ranking selection was partially solved. This selective encoding approach could significantly outperform all the other methods including the state-of-the-art encodings of Rintanen.

4. Improving Plans

With intelligent systems becoming ubiquitous there is a need for planning systems to operate in almost real-time. Sometimes it is necessary to provide a solution in a very little time to avoid imminent danger (e.g damaging a robot) and prevent significant financial losses. Satisficing planning engines such as FF [19], Fast Downward [18] or LPG [16] are often able to solve a given problem quickly, however, quality of solutions might be low. Optimal planning engines, which guarantee the best quality solutions, often struggle even on simple problems. Therefore, a reasonable way how to improve the quality of the solutions produced by satisficing planning engines is to use post-planning optimization techniques.

In the thesis we review various techniques that have been proposed for post-planning plan optimization [31, 24, 13, 3, 30] and introduce some new ones. We restrict ourselves to a specific sub-category of post-planning plan optimization – determining and removing redundant actions from plans. An influential work [15] defines different categories of redundant actions and provides some complexity results, in particular, that removing all redundant actions is an NP-hard optimization problem.

There are several heuristic approaches [11, 15, 24], which can identify most of the redundant actions in plans in polynomial time. One of the most efficient of these approaches was introduced in [15] under the name Linear Greedy Justification. It was reinvented in [24] and called Action Elimination. In the thesis we use the latter name and extend the algorithm to take into account the action costs. Our new algorithm is called Greedy Action Elimination and it works by finding all the polynomially detectable subsequences of redundant actions and then removing the most costly subsequence. The worst-case time complexity of Greedy Action Elimination is $O(n^3 p)$, where $n = |P|$ (the length of the original plan) and p is the maximum number of preconditions or effects any action in P has.

4.1 SAT and MaxSAT Reduction

Our goal is to encode a given planning task Π and a valid plan P for Π into a CNF formula $F_{\Pi,P}$, such that each satisfying assignment of $F_{\Pi,P}$ represents a plan reduction P' of P, i.e., a sub-sequence of P which is also a valid plan for Π.

The main idea of the translation is to encode the fact, that if a certain condition c_i is required to be true at some time i in the plan, then one of the following must hold:

- The condition c_i is true since the initial state and there is no opposing action of c_i (an action that destroys c_i) with a rank smaller than i.

- There is a supporting action $P[j]$ of c_i (an action that sets up c_i) with the rank $j < i$ and there is no opposing action of c_i with its rank between j and i.

Using the encoding of the condition requirement it is easy to encode the dependencies of the actions (their preconditions) from the input plan and the goal conditions of the problem.

In the thesis we describe in detail the Boolean variables and clauses of $F_{\Pi,P}$, we compute and upper bound on its size and give a proof of the correctness of the encoding.

The formula $F_{\Pi,P}$ is then used to construct the following three translation based methods for removing redundant actions from plans.

- *SAT-based Reduction.* By adding a clause to $F_{\Pi,P}$ that represents the condition, that at least one action must be removed from P, we obtain a formula that is satisfiable if and only if P is a redundant plan. By iteratively using this encoding and a SAT solver we can eliminate redundant actions from P until we achieve a perfectly justified (irredundant) plan. No more actions can be removed from the resulting plan, nevertheless, it might be the case, that if we had removed a different set of redundant actions from the initial plan P, we could have arrived at a shorter perfectly justified plan. In other words, the elimination of redundancy is not confluent, i.e., the result depends on the order in which the redundant actions are removed. This issue is addressed by the following two methods.

- *Partial MaxSAT-based Reduction.* A partial MaxSAT formula consists of hard and soft clauses. Let $F_{\Pi,P}$ be the hard clauses of a formula F_{MAX} and unit clauses representing the removal of each action be the soft clauses of F_{MAX}. Then a partial MaxSAT solver will find an assignment ϕ for F_{MAX} that satisfies all the hard clauses (which enforces the validity of the plan reduction) and satisfies as many soft clauses as possible (which removes as many actions as possible). In this way we can remove the maximum number of redundant actions from a given plan in a single call of a partial MaxSAT solver. The obtained plan is referred to as the *Minimal Length Reduction* (MLR), therefore we will call this method MLR.

- *Weighted Partial MaxSAT-based Reduction.* In the case, that the actions in a plan have different costs, it might be desirable to remove redundant actions of the highest total cost rather than removing the highest number of them. This can be achieved by adding weights to the unit soft clauses of F_{MAX} that are equal to the cost of the corresponding actions and using a weighted partial MaxSAT solver. The obtained plan is called *Minimal Reduction* (MR), therefore we will refer to this method as MR.

4.2 Experiments

We implemented all the above mentioned plan reduction algorithms and compared them on plans obtained by three state-of-the-art satisficing planners (Madagascar, Fast Downward, and Metric FF) for the problems of the 2011 International

Table 4.1: The cost of removed actions. The table contains the number of found plans (#P) out of 20 in each domain, their total cost and the total cost of eliminated actions by the six redundancy elimination methods. IAE is a heuristic method for removing pairs of inverse actions, MLR is the partial MaxSAT-based method, and MR is the weighted partial MaxSAT-based method.

	Domain	#P	Cost	IAE	AE	GAE	SAT	MLR	MR
Metric FF	elevators	20	25618	2842	2842	2842	2842	2842	2842
	floortile	2	195	29	30	30	30	30	30
	nomystery	5	107	0	0	0	0	0	0
	parking	18	1546	118	124	124	124	124	124
	pegsol	20	300	0	0	0	0	0	0
	scanalyzer	18	1137	0	62	62	62	62	62
	sokoban	13	608	0	2	2	2	2	2
	transport	6	29674	2650	3013	3035	3013	3035	3035
Fast Downward	barman	20	7763	436	753	780	893	926	926
	elevators	20	28127	1068	1218	1218	1218	1218	1218
	floortile	5	572	66	66	66	66	66	66
	nomystery	13	451	0	0	0	0	0	0
	parking	20	1494	4	4	4	4	4	4
	pegsol	20	307	0	0	0	0	0	0
	scanalyzer	20	1785	0	78	78	78	78	78
	sokoban	17	1239	0	58	58	102	102	102
	transport	17	74960	4194	5259	5260	5259	5260	5260
Madagascar	barman	8	3360	296	591	598	591	606	606
	elevators	20	117641	7014	24096	24728	26702	28865	28933
	floortile	20	4438	96	96	96	96	96	96
	nomystery	15	480	0	0	0	0	0	0
	parking	18	1663	152	152	152	152	152	152
	pegsol	19	280	0	0	0	0	0	0
	scanalyzer	18	1875	0	232	236	232	236	236
	sokoban	1	33	0	0	0	0	0	0
	transport	4	20496	4222	6928	7507	7444	7736	7736

Planning Competition [12]. All the experiments were run on a computer with Intel Core i7 960 CPU @ 3.20 GHz processor and 24 GB of memory. The planners had a time limit of 10 minutes to find the initial plans. The runtime for the optimization was unlimited, however it never took more than 5 minutes for any problem. We used Sat4j [7] for SAT solving, QMaxSAT [23] for partial MaxSAT solving and Toysat [29] for weighted partial MaxSAT solving.

The total cost of the removed actions is displayed in Table 4.1. We can observe, that the IAE method is the weakest followed by AE and GAE. The AE algorithm, although it ignores the action costs, performs rather well. Except for 8 planner/domain combinations it achieves minimal reduction, i.e., the best possible result. The GAE algorithm improves upon AE in 7 cases and achieves minimal reduction in all but 5 planner/domain pairs.

The partial MaxSAT-based method (MLR) is guaranteed to remove the maximum number of redundant actions (not considering their cost) and this is also enough to achieve minimal plan reduction in each case except for the Madagascar plans for the elevators domain. As expected, the weighted partial MaxSAT-based method (MR) provides the best results, however these results are often not strictly better than the results of the polynomial methods.

In the thesis we provide further experimental results regarding the total number of removed redundant actions, the runtime of the methods as well as the number of instances, where the heuristic methods achieved perfect justification (irredundant plans).

Clearly, the MR method is guaranteed to provide minimal reduction of plans and therefore cannot be outperformed (in terms of quality) by the other methods. Similarly, the MLR method cannot be outperformed in terms of plan lengths. Despite the exponential worst-case time complexity of these methods, runtimes are usually very low and in many cases even lower than the other polynomial methods we compared with. On the other hand, when the problem becomes harder the runtimes can significantly increase. We have observed that the problem of determining redundant actions (including minimal reduction) is in most of the cases very easy. Therefore, the measured runtimes often depend more on the efficiency of implementation of particular methods rather than the worst-case complexity properties.

Our results also showed that in the most cases using the polynomial method (AE or GAE) provides minimal reduction, so the MR method usually does not lead to strictly better results. Guaranteeing in which cases (Greedy) AE provides minimal reduction is an interesting open question.

5. Conclusion

In the thesis we have shown how can satisfiability (SAT) and maximum satisfiability (MaxSAT) solvers be efficiently used to both find plans and improve them. Finding plans via SAT solving is not a new idea. It has been around for several decades and it is one of the most successful approaches to automated planning.

Our main contribution to the topic of planning as SAT is the introduction of two new encoding schemes, the Reinforced and the $R^2\exists$-Step encoding. These two encodings work well for different sets of planning problems (domains) but we were able to find a simple rule which allows automatic selection of the best encoding for a given planning task. Using this rule we designed a combined encoding that can significantly outperform the existing state-of-the-art encodings.

As for the second problem – the improvement of plans, we have focused on the special case of removing redundant (unnecessary) actions from plans, which is an NP-hard optimization problem. Prior to our work, there existed only heuristic algorithms that are not guaranteed to remove all the redundant actions. The most successful of these algorithms is called Action Elimination (AE). Based on the ideas of AE we have introduced our own heuristic algorithm – Greedy Action Elimination (GAE), which, contrary to AE, takes actions cost into account. GAE outperformed AE and the other existing heuristic approaches on benchmark problems.

Furthermore, we have introduced a SAT encoding for the problem of plan redundancy. Using this encoding we have proposed three new methods which can completely solve three optimization problems related to redundancy elimination. The first method uses a SAT solver to produce perfectly justified plans, i.e., plans without redundant actions. The second method uses a partial MaxSAT solver to remove the highest possible number of redundant actions from plans. Finally, the third method guarantees to remove the set of redundant actions with the highest total cost and uses a weighted partial MaxSAT solver. Thanks to the existence of powerful SAT and MaxSAT solvers, these methods work very well in practice with the current state-of-the-art planners and benchmark problems.

Bibliography

[1] Christer Bäckström and Bernhard Nebel. Complexity results for sas+ planning. *Computational Intelligence*, 11:625–656, 1995.

[2] Tomáš Balyo. Relaxing the relaxed exist-step parallel planning semantics. In *ICTAI*, pages 865–871. IEEE, 2013.

[3] Tomáš Balyo, Roman Barták, and Pavel Surynek. Shortening plans by local re-planning. In *ICTAI*, pages 1022–1028. IEEE, 2012.

[4] Tomáš Balyo, Roman Barták, and Daniel Toropila. Two semantics for step-parallel planning: Which one to choose? *Proceedings of the 29th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2011)*, pages 9–15, 2011.

[5] Tomáš Balyo and Lukáš Chrpa. Eliminating all redundant actions from plans using sat and maxsat. *Proceedings of Knowledge Engineering for Planning and Scheduling (KEPS)*, 2011.

[6] Tomáš Balyo, Lukáš Chrpa, and Asma Kilani. On different strategies for eliminating redundant actions from plans. *Proceedings of The Seventh Annual Symposium on Combinatorial Search (SOCS), To appear*, 2014.

[7] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.

[8] Armin Biere. Lingeling and plingeling home page. http://fmv.jku.at/lingeling/, May 2014.

[9] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[10] Tom Bylander. The computational complexity of propositional strips planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[11] Lukáš Chrpa, Thomas Leo McCluskey, and Hugh Osborne. Determining redundant actions in sequential plans. In *Proceedings of ICTAI*, pages 484–491, 2012.

[12] Amanda Jane Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez Celorrio, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1), 2012.

[13] Sam J. Estrem and Kurt D. Krebsbach. Airs: Anytime iterative refinement of a solution. In *Proceedings of FLAIRS*, pages 26–31, 2012.

[14] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[15] Eugene Fink and Qiang Yang. Formalizing plan justifications. In *In Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*, pages 9–14, 1992.

[16] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Planning through stochastic local search and temporal action graphs. *Journal of Artificial Intelligence Research (JAIR)*, 20:239 – 290, 2003.

[17] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning: Theory and Practice.* Morgan Kaufmann Publishers, 2004.

[18] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research (JAIR)*, 26:191–246, 2006.

[19] Joerg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[20] Ruoyun Huang, Yixin Chen, and Weixiong Zhang. A novel transition based encoding scheme for planning as satisfiability. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[21] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI 92: Tenth European Conference on Artificial Intelligence*, pages 359–363, Vienna, Austria, 1992.

[22] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *ECAI*, pages 359–363, 1992.

[23] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. Qmaxsat: A partial max-sat solver. *JSAT*, 8(1/2):95–100, 2012.

[24] Hootan Nakhost and Martin Müller. Action elimination and plan neighborhood graph search: Two algorithms for plan improvement. In Ronen I. Brafman, Hector Geffner, Jörg Hoffmann, and Henry A. Kautz, editors, *ICAPS*, pages 121–128. AAAI, 2010.

[25] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artif. Intell.*, 193:45–86, 2012.

[26] Jussi Rintanen. Planning as satisfiability: state of the art. http://users.cecs.anu.edu.au/ jussi/satplan.html, July 2013.

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

[28] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Sat-based parallel planning using a split representation of actions. In Alfonso Gerevini, Adele E. Howe, Amedeo Cesta, and Ioannis Refanidis, editors, *ICAPS*. AAAI, 2009.

[29] Masahiro Sakai. Toysolver home page. https://github.com/msakai/toysolver, May 2014.

[30] Fazlul Hasan Siddiqui and Patrik Haslum. Plan quality optimisation via block decomposition. In Francesca Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013.

[31] Carl Henrik Westerberg and John Levine. Optimising plans using genetic programming. In *Proceedings of ECP*, pages 423–428, 2001.

List of Publications

- Tomáš Balyo, Roman Barták, Otakar Trunda: Reinforced Encoding for Planning as SAT, Student Conference on Planning in Artificial Intelligence and Robotics (PAIR) 2014
- Tomáš Balyo, Lukáš Chrpa, Asma Kilani: On Different Strategies for Eliminating Redundant Actions from Plans, The Seventh Annual Symposium on Combinatorial Search (SOCS) 2014
- Tomáš Balyo, Andreas Froehlich, Marijn Heule, Armin Biere: Everything You Always Wanted to Know About Blocked Sets (But Were Afraid to Ask), The 17th International Conference on Theory and Applications of Satisfiability Testing (SAT) 2014
- Tomáš Balyo, Lukáš Chrpa: Eliminating All Redundant Actions from Plans Using SAT and MaxSAT, The 5th Workshop on Knowledge Engineering for Planning and Scheduling (KEPS) 2014
- Tomáš Balyo: Relaxing the Relaxed Exist-Step Parallel Planning Semantics, The 24th International Conference on Tools with Artificial Intelligence (ICTAI) 2013
- Martin Babka, Tomáš Balyo, Jaroslav Keznikl: Solving SMT Problems with a Costly Decision Procedure by Finding Minimum Satisfying Assignments of Boolean Formulas, Studies in Computational Intelligence, volume 496, 2013
- Martin Babka, Tomáš Balyo, Jaroslav Keznikl: Finding Minimum Satisfying Assignments of Boolean Formulas, The 26th International FLAIRS Conference 2013
- Martin Babka, Tomáš Balyo, Ondrej Čepek, Štefan Gurský, Petr Kučera, Václav Vlček: Complexity issues related to propagation completeness, Artificiall Intellingence, volume 206, 2013
- Tomáš Balyo, Roman Barták, Pavel Surynek: Shortening Plans by Local Re-Planning, The 24th International Conference on Tools with Artificial Intelligence (ICTAI) 2012
- Tomáš Balyo, Roman Barták, Pavel Surynek: On Improving Plan Quality via Local Enhancements, The Fifth Annual Symposium on Combinatorial Search (SOCS) 2012
- Tomáš Balyo, Štefan Gurský, Petr Kučera, Václav Vlček: On Hierarchies over the SLUR Class, International Symposium on Artificial Intelligence and Mathematics (ISAIM) 2012

- Tomáš Balyo, Dan Toropila, Roman Barták: Two Semantics for Step-Parallel Planning: Which One to Choose?, The 29th Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG) 2011
- Tomáš Balyo: Decomposing Boolean formulas into connected components, The 20th Week of Doctoral Students (WDS) 2011
- Tomáš Balyo, Pavel Surynek: Efektivní heuristika pro SAT založená na znalosti komponent souvislosti grafu problému (An Efficient Heuristic for SAT Exploiting Connected Components of the Problem), Znalosti 2009