Satisfiability Planning with Constraints on the Number of Actions

Markus Büttner and **Jussi Rintanen**[*]
Albert-Ludwigs-Universität Freiburg, Institut für Informatik
Georges-Köhler-Allee, 79110 Freiburg im Breisgau
Germany

Abstract

We investigate satisfiability planning with restrictions on the number of actions in a plan. Earlier work has considered encodings of sequential plans for which a plan with the minimal number of time steps also has the minimum number of actions, and parallel (partially ordered) plans in which the number of actions may be much higher than the number of time steps. For a given problem instance finding a parallel plan may be much faster than finding a corresponding sequential plan but there is also the possibility that the parallel plan contains unnecessary actions.

Our work attempts to combine the advantages of parallel and sequential plans by efficiently finding parallel plans with as few actions as possible. We propose techniques for encoding parallel plans with constraints on the number of actions. Then we give algorithms for finding a plan that is optimal with respect to a given number of steps and an anytime algorithm for successively finding better and better plans. We show that as long as guaranteed optimality is not required, our encodings for parallel plans are often much more efficient in finding plans of good quality than a sequential encoding.

Introduction

We investigate the encoding of parallel plans in the propositional logic with an upper bound on the number of actions. The main application of this work is the improvement of parallel plans which may contain more actions than necessary.

The basic idea of planning as satisfiability is simple. For a given number n of steps a propositional formula Φ_n is produced. This formula Φ_n is satisfiable if and only if a plan with n steps exists. From a satisfying assignment for Φ_n a solution plan of n steps can be easily generated.

Efficiency of satisfiability planning can be improved by allowing parallelism in plans. This means that under certain conditions several actions may be taken at the same step (Kautz & Selman 1996; Rintanen, Heljanko, & Niemelä 2005). For many planning problems this gives a great speedup since parallelism reduces the number of steps needed and hence also the size of the propositional formulae. On the other hand, the price for the speed-up is the loss of optimality. A parallel plan with the minimum number of steps may have many more actions than an optimal plan. As an example consider a logistics problem where a truck can be moved

[*]Research project partially funded by DFG grant RI 1177/2-1.

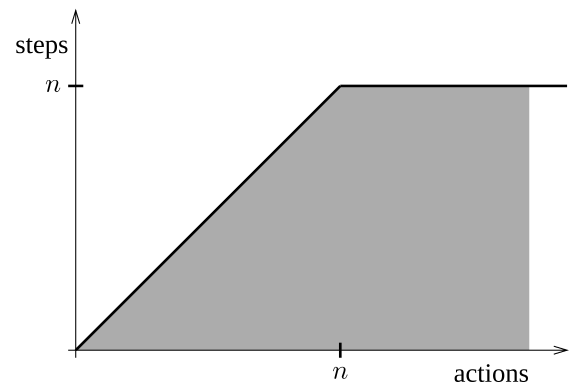

Figure 1: Numbers of steps and actions in sequential and parallel plans

between locations A and B. In a parallel plan the truck might be pointlessly moved between the two locations while other useful actions are taken in parallel. The fundamental reason for the unnecessary actions is that satisfiability algorithms do not attempt to find satisfying assignments with few true propositions and might for example arbitrarily assign value true to propositions that are irrelevant for the satisfiability of the formula in question.

If we want to find an optimal plan we are in general forced to consider sequential plans instead of parallel plans. Knowing that there are no plans with less than n steps and no plans with n steps and less than m actions does not mean that there are no plans with more than n steps and less than m actions. Ultimately, to show that no plans with less than m actions exist we must show that no sequential plans with $m-1$ steps (and hence also $m-1$ actions) exist.

To illustrate the difference between sequential and parallel plans consider Figure 1. Formulae for sequential planning are parameterized with n that denotes both the number of steps and the number of actions in a plan, as only one action is allowed at each step. This corresponds to the diagonal line in Figure 1. Formulae for parallel planning are parameterized with the number n of non-empty steps. Because each step contains one or more actions, there is no explicit upper bound on the number of actions in a plan with n steps, and each plan with n steps has at least n actions.

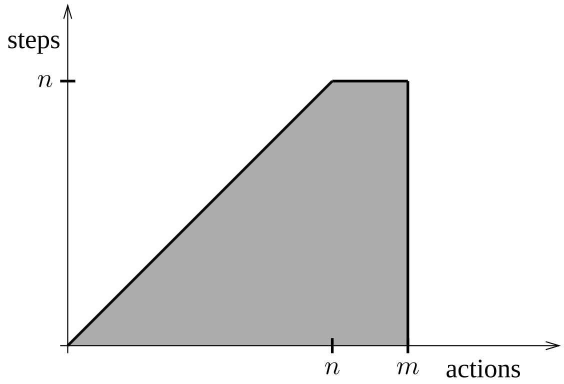

Figure 2: Restrictions on both the number of steps and the number of actions

The possible plans for parameter n therefore correspond to the shaded region in Figure 1.

Our objective in this work is to develop techniques for improving the quality of parallel plans without having to consider sequential plans. The work is based on setting an upper bound both on the number of steps and the number of actions in plans. The formulae we produce correspond to the question: Does a plan with at most n steps and at most m actions exist? Possible solutions have $x \leq n$ nonempty steps and $y \leq m$ actions, corresponding to the shaded region in Figure 2. The difference to earlier works on satisfiability planning is that our formulae are also parameterized by the number of actions m.

Also other popular approaches to solving planning problems – including heuristic search algorithms (Bonet & Geffner 2001) – yield techniques for finding both arbitrary plans and plans that are guaranteed to contain the minimum possible number of actions. With heuristic search optimal plans can be found by using systematic optimal heuristic search algorithms like A$*$ or IDA$*$ together with admissible heuristics (Haslum & Geffner 2000). Plans without optimality guarantees can be often found more efficiently by using local search algorithms and non-admissible heuristics (Bonet & Geffner 2001).

The structure of this paper is as follows. We first give a brief overview of planning as satisfiability and the notions of parallel plans used. Then we describe encodings of *cardinality constraints* in the propositional logic, that is constraints on the number of true propositions in satisfying assignments of a propositional formula. This is the basis of restricting the number of actions in parallel plans. To keep the size of the formulae for cardinality constraints small we propose a technique for reducing the number of propositions need to be counted. Then we propose two planning algorithms based on our encoding of parallel plans with constraints on the number of actions. Finally we give results of experiments that illustrate the differences in the efficiency of sequential and parallel encodings of planning as satisfiability and runtime behavior of SAT solvers on different combinations of restrictions on the number of steps and actions in parallel plans.

Planning as Satisfiability

Planning as satisfiability was first proposed by Kautz and Selman (1992). The notion of plans initially was sequential: at every step in a plan exactly one action is taken. After the success of the GraphPlan algorithm that used parallel plans (Blum & Furst 1997), Kautz and Selman also proposed parallel encodings of planning (Kautz & Selman 1996). In these works two or more actions can be taken in parallel at the same step as long as the actions are *independent*. Independence is a syntactic condition that guarantees that the parallel actions can be put to any total order and a valid plan is obtained in every case. Recently, the possibility of more parallelism than what is allowed by independence has been utilized by Rintanen et al. (2004) based on ideas by Dimopoulos et al. (1997).

The basic idea in satisfiability planning is to encode the question of whether a plan with n steps exists as a propositional formula of the following form.

$$\Phi_n = I_0 \wedge \left(\bigwedge_{i \in \{0,\ldots,n-1\}} R(A_i, P_i, P_{i+1}) \right) \wedge G_n$$

The propositional variables in this formula are $P_0 \cup \cdots \cup P_n \cup A_0 \cup \cdots \cup A_{n-1}$ expressing the values of state variables at time points $0, \ldots, n$ and which actions are are taken at the steps $0, \ldots, n-1$.

The formula I_0 describes the unique initial state: it is a formula over the propositional variables P_0 for time point 0. Similarly G_n is a formula describing the goal states in terms of propositional variables for time point n.

The transition relation formula $R(A_t, P_t, P_{t+1})$ describes how the values of the state variables change when some of the actions are taken. The propositional variables P_t are for the state variables at time t and the propositional P_{t+1} for time $t + 1$. There is one variable in A_t for every action. If an action is taken at time point t, the corresponding propositional variable in A_t is true.

If the formula Φ_n is satisfiable then a plan with n steps exists. Plan search now proceeds by generating formulae Φ_n for different values of $n \geq 0$, testing their satisfiability, and constructing a plan from the first satisfying assignment that is found.[1] The propositional variables in $A_0 \cup \cdots \cup A_{n-1}$ directly indicate which actions are taken at which step.

The way the transition relation formulae $R(A_t, P_t, P_{t+1})$ are defined depends on the notion of plans used and other encoding details. For different possibilities see (Kautz & Selman 1996; Ernst, Millstein, & Weld 1997; Rintanen, Heljanko, & Niemelä 2005). For our purposes in this paper it is sufficient that we can determine which actions are taken by looking at the values of the propositional variables in $A_0 \cup \cdots \cup A_{n-1}$. The basic question we want to answer is whether a plan with n steps and at most m actions exists. For this purpose we use the formula Φ_n together with an encoding of the cardinality constraint that says that at most m of the propositional variables in $A_0 \cup \cdots \cup A_{n-1}$ may be true. How these cardinality constraints can be encoded is described in the next section.

[1] See Rintanen (2004a) for other approaches.

Constraints on the Number of Actions

Let A be the set of actions in our problem instance. If we are looking for a plan with n steps we have $k = |A|n$ propositions that correspond to the possible actions at the n steps. We call these propositions *the indicator propositions* since they indicate that an action is taken, and denote them by $i_1, \ldots, i_k$. In restricting the number of actions in a plan we have to encode the cardinality constraint that the set of true indicator propositions in an assignment has a cardinality of at most m.

We have implemented two different encodings of cardinality constraints. The first encoding is based on an injective mapping of the indicator propositions to integers $1, \ldots, m$. A proposition $p_{x,y}$ with $x \in \{1, \ldots, k\}$ and $y \in \{1, \ldots, m\}$ indicates that i_x is mapped to the integer y. Additional formulae make sure that *a)* every indicator proposition is mapped to at least one integer (this can be encoded with k clauses with m literals in each), and *b)* the mapping is injective (needs $\frac{km-m}{2}$ clauses with 2 literals in each), that is, no two indicator propositions are mapped to the same integer. The total size of the clauses (in terms of number of literal occurrences) is $km + km - m = 2km - m$. Our initial experiments with this encoding did lead to rather good results. An interesting result was that an upper bound on actions often lead to a plan with many fewer actions than the upper bound required or even the minimum number.

Then we implemented a variant of the encoding of cardinality constraints developed by Bailleux and Boufkhad (2003). This encoding, although slightly bigger, turned out to be even more efficient than the one based on injections. This encoding uses a binary tree in which the leaves are the indicator propositions. Every internal node N of the tree corresponds to an integer in a range $\{0, \ldots, r_N\}$ indicating how many of the leaves in that subtree are true, represented in unary by n propositions $p_{N,1}, \ldots, p_{N,r_N}$. Here r_N is the minimum of $m + 1$ and the total number of leaves in the subtree.[2] A proposition $p_{N,i}$ is true if the number of true indicator propositions in the tree is $\geq i$. The value of an internal node is the sum of its two child nodes. The tree is recursively constructed as follows.

1. Each leaf node (corresponding to one indicator proposition) is interpreted as an integer in $\{0, 1\}$ where false corresponds to 0 and true to 1. We denote the indicator proposition of a leaf node N by $p_{N,1}$.
2. An internal node N with two children nodes X and Y has the range $r_N = [0, \ldots, \min(m+1, r_X + r_Y)]$). The value of node N is the sum of the value of X and Y and hence it remains to encode this addition. This is done by formulae $(p_{X,j_1} \wedge p_{Y,j_2}) \rightarrow p_{N,j_1+j_2}$ for all $j_1 \in \{1, \ldots, r_X\}$ and $j_2 \in \{1, \ldots, r_Y\}$ such that $j_1 + j_2 \leq r_N$, and formulae $p_{X,j_1} \rightarrow p_{N,j_1}$ and $p_{Y,j_2} \rightarrow p_{N,j_2}$ for all $j_1 \in \{1, \ldots, r_X\}$ and $j_2 \in \{1, \ldots, r_Y\}$. For this at most $mm + m + m = m^2 + 2m$ clauses are needed.

[2] Bailleux and Boufkhad (2003) always count up to k, but because our cardinality constraint m is often very small in comparison to k, counting further than $m + 1$ increases the formula sizes unnecessarily.

From the propositions for the root node N_0 we can determine whether the number of true indicator propositions exceeds m: the formula $\neg p_{N_0,m+1}$ is true if and only if the number of true indicator propositions is at most m.

Bailleux and Boufkhad (2003) have shown that this encoding is more efficient that encodings that more compactly represent counting for example by using a binary encoding. The efficiency seems to be due to the possibility of determining violation of the cardinality constraints by means of unit resolution already when only a subset of the indicator propositions have been assigned a value.

Assuming that k is a power of 2, the tree has $k - 1$ inner nodes and depth $\log_2 k$. Hence an upper bound on the number of clauses for representing the cardinality constraint is $(k - 1)(m^2 + 2m)$.

Cardinality constraints have many applications. Bailleux and Boufkhad (2003) discuss the discrete tomography problem, a simplified variant of the tomography problems having applications in medicine. The computation of medians of sequences of binary numbers in digital signal processing also involves cardinality constraints and further encodings have been developed for that purpose (Valmari 1996).

Unparallelizable Sets of Actions

To reduce the sizes of the formulae encoding the cardinality constraints we consider possibilities of counting a smaller number of indicator propositions than $k = |A|n$ for plans with n steps. A smaller number of indicator propositions is often possible because there are dependencies between actions, making it impossible to take some actions in parallel.

Our idea is the following. We first partition the set of all actions into x subsets so that taking one action from any set makes it impossible to take the remaining actions in the set in parallel. Then we introduce a new indicator proposition for each of these sets. This proposition indicates whether an action of this subset is taken. Now we need to count only xn indicator propositions instead of $|A|n$. In some cases x is much smaller than $|A|$.

Assume that we have a set $S = \{a_1, a_2, \ldots, a_q\}$ of actions of which at most one can be taken at one step. In the following formulae we will use the symbols a_x, $x \in \{1, \ldots, q\}$ to denote that action a_x is taken. Then we introduce a new proposition i_S and the following formulae.

$$a_1 \rightarrow i_S, a_2 \rightarrow i_S, \ldots, a_q \rightarrow i_S$$

Obviously, if one of the actions a_1 to a_q is taken, i_S is true. Since i_S indicates that one action of the set S is taken, we call i_S the indicator proposition of the set S.

It remains now to partition the set of all actions into as few sets of actions as possible. The identification of sets of actions of which at most one can be taken at a time is equivalent to the graph-theoretic problem of finding *cliques* of a graph, that is, complete subgraphs in which there is an edge between every two nodes. In the graph $\langle N, E\rangle$ the set N of nodes consists of the actions and there is an edge $(a, a') \in E$ between two nodes if the actions cannot be taken at the same time.

Identification of big cliques may be very expensive. Testing whether there is a clique of size n is NP-complete (Karp

function PartitionToCliques(A, E)
if $(A \times A) \cap E = \emptyset$ **then return** $\{\{a\}|a \in A\}$;
Divide A into three sets A_+, A_- and A_0
so that $(A_+ \times A_-) \subseteq E$
and $|A_+| > 0$ and $|A_-| > 0$;
P_+ := PartitionToCliques(A_+, E);
P_- := PartitionToCliques(A_-, E);
P_0 := PartitionToCliques(A_0, E);
Let $P_+ = \{p_1^+, \ldots, p_i^+\}$;
Let $P_- = \{p_1^-, \ldots, p_j^-\}$;
Without loss of generality assume that $i \geq j$;
$P := \{p_1^+ \cup p_1^-, \ldots, p_j^+ \cup p_j^-\} \cup P_0 \cup \{p_{j+1}^+, \ldots p_i^+\}$;
return P;

Figure 3: An algorithm for partitioning the set of actions

1972; Garey & Johnson 1979). Hence we cannot expect to have a polynomial-time algorithm that is guaranteed to find the biggest cliques of a graph. Approximation algorithms for locating cliques exist (Hochbaum 1998).

We have used a simple polynomial-time algorithm for identifying sets of actions that cannot be taken in parallel. We construct a graph where the nodes are the actions. Two nodes a and a' are connected by an edge if it is not possible to take the action a and a' in parallel. The problem is to find a partition of the graph into as few cliques as possible. The recursive algorithm given in Figure 3 takes as input the set A of actions and returns a partition of A into cliques.

In the base case of recursion the actions in A have no dependencies and therefore form an independent set ($(A \times A) \cap E = \emptyset$) of the graph. Each member of A now forms a trivial 1-element clique, and these cliques are returned.

In the recursive case a complete bipartite subgraph A_+, A_- is identified (heuristically, see below for details), and the both partitions A_+ and A_- as well as the remaining nodes A_0 are recursively partitioned to cliques. As a final step cliques p^+ and p^- respectively corresponding to A_+ and A_- can be pairwise unioned to obtain bigger cliques $p^+ \cup p^-$ (because $A_+ \times A_- \subseteq E$) that are returned together with the cliques obtained from A_0.

The partitioning of the set A into the sets A_+, A_- and A_0 is done as follows. We choose a state variable X and define the following sets.

1. A_+ consists of all actions a for which X is necessarily be true when action a is taken. This means that X is a logical consequence of the precondition of a or there is an invariant $l \vee X$ (we use the invariants computed by the algorithm of Rintanen (1998)) and the precondition of a entails the complement $\bar{l}$ of the literal l.
2. A_- consists of all actions a for which X is necessarily be false when a is taken.
3. $A_0 = A - (A^+ \cup A^-)$.

Now there is an edge between any two nodes respectively from A_+ and A_-. For commonly used benchmark problems this algorithm works well and returns many non-trivial cliques, reducing the number of indicator propositions.

procedure Algorithm1(**int** n)
P := any parallel plan with n steps;
repeat
m := the number of actions in P;
P_{best} := P;
try to find a plan P with n steps and $\leq m - 1$ actions;
until no plan found;
return P_{best};

Figure 4: Algorithm 1

The algorithm runs in polynomial time with respect to the number of actions $|A|$. All components of the function are polynomial and the number of recursive calls is polynomial.

Lemma 1. *Given a set of $|A| > 0$ actions the function PartitionToCliques(A, E) makes at most $3(|A| - 1)$ recursive calls. For $|A| = 0$ it makes no recursive calls.*

Proof. We prove the statement by induction over the size $|A|$ of the input set.

Base case: If the input set A is empty or has only one member the function returns without recursive calls. The statement is hence true for $|A| = 1$ and $|A| = 0$.

Inductive case: If $|A| > 1$ the function makes at most $3 + 3(|A_+| - 1) + 3(|A_-| - 1) + \max\{0, 3(|A_0| - 1)\}$ recursive calls. Note that the sets A_+ and A_- are non-empty and $|A| = |A_+| + |A_-| + |A_0|$. It follows that the function makes at most $3(|A| - 1)$ recursive calls. $\square$

Planning Algorithms

The restriction on the number of actions in a plan can be used in planning algorithms in several ways. We consider the computation of plans of a given number of steps and a minimum number of actions, as well as an anytime algorithm that is eventually guaranteed to find a plan with the smallest possible number of actions.

Optimal Plans of a Given Length

For this application we first find any parallel plan of length n without any restrictions on the number of actions. Let m be the number of actions in this plan. To find a plan with n steps and the the smallest possible number of actions we simply compute a plan for length n with the cardinality constraint that restricts the number of actions to at most $m - 1$. If such a plan does not exist then m is already the minimum number of actions a plan of length n can have. Otherwise we get a better plan with some number $m_1 \leq m - 1$ of actions. This is repeated until no plans with a smaller number of actions is found. Finally we get a plan with the smallest number of actions with respect to the plan length n. The algorithm is given in Figure 4.

Since proving the optimality, that is, performing the last unsatisfiability test, may be much more expensive than the previous tests, if we are just interested in finding a good plan without guaranteed optimality we suggest terminating the computation after some fixed amount of time.

```
procedure Algorithm2()
P := any parallel plan with a minimal number of steps;
m := the number of actions in P;
n := the number of steps in P;
while m < n do
begin
  if plan P with n steps and ≤ m − 1 actions exists
  then
    begin
      P_best := P;
      m := the number of actions in P_best;
    end
  else n := n + 1;
end
return P_best;
```

Figure 5: Algorithm 2

Anytime Algorithm for Finding Good Plans

When we have to find a plan with the smallest possible number of actions the use of an encoding for sequential plans is unavoidable. Assume we have found a plan with n steps and m actions. We could now show that there are no plans with n steps and less than m actions for some $n < m$. However, there might still be plans that have less than m actions and more than n steps. So we are always ultimately forced to test for the existence of plans with $m - 1$ steps and $m - 1$ actions. Hence the problem of optimal planning always reduces to sequential planning and parallel plans are of no use.

However, finding plans with guaranteed optimality may be too expensive and our interest might be simply to find plans of good quality without guarantees for optimality. For this purpose the use of parallel plans is more promising because they can often be found much more efficiently than corresponding sequential plans.

We extend Algorithm 1 to repeatedly find plans with fewer and fewer actions, increasing the number of steps when necessary. Algorithm 2 is given in Figure 5.

This algorithm first minimizes the number of actions for a given number of steps. If a plan with the smallest number m of actions has been found the number of steps n is increased and an attempt to find a plan with $m - 1$ actions or less is repeated. This decreasing of the number of actions and increasing the number of steps is alternated until the algorithm has proved that no further reduction in the number of actions is possible. On the last iteration the algorithm has to prove that no plans of $m - 1$ actions and $m - 1$ steps exist, which is a question answered by sequential planning.

As our initial motivation was to avoid using an encoding for sequential plans, this algorithm should be used with restrictions on its runtime so that the limit case is not reached. Essentially, this is an anytime algorithm that yields the better plans the longer it is run.

Experiments

In this section we present results from some experiments which were run on a 3.6 GHz Intel Xeon processor with 512 KB internal cache using the satisfiability program Siege (version 4). Siege runtimes vary because of randomization, and the runtimes we give are averages of 20 runs. The problem instances are from the AIPS planning competitions. The logistic problems are from the year 2000. A dash in the runtime tables means a missing upper bound for the runtime when a satisfiability test did not finish in 10 minutes. All runtimes are in seconds.

problem	steps	actions	min
Logistics4_0	6	26	20
Logistics5_0	6	29	27
Logistics6_0	6	32	25
Logistics7_0	7	44	38
Logistics8_0	7	39	31
Logistics9_0	7	43	36
Logistics10_0	9	79	46
Driverlog2	8	24	19
Driverlog3	5	14	12
Driverlog4	5	22	18
Driverlog5	5	22	18
Driverlog6	5	21	11
Driverlog7	5	19	13
Driverlog8	6	29	25
Driverlog9	9	28	22
Zenotravel4	4	11	8
Zenotravel5	4	12	11
Zenotravel6	3	13	12
Depots4	14	56	30
Depots5	18	66	49
Depots6	22	90	64
Satellite7	4	33	27
Satellite10	5	55	34
OpticalTelegraph1	13	36	35
Mystery9	4	11	8
Movie6	2	61	7
Movie10	2	84	7
Mprime2	4	25	9
Mprime3	3	17	4

Table 1: Number of actions found by a satisfiability planner without restrictions on number of actions versus the smallest number of actions for the given number of steps

The parallel plans were found by the most efficient 1-linearization encoding considered by Rintanen et al. (2004) and the sequential plans by an encoding that differs from the parallel one only with respect to the constraints on parallel actions: at each step only one action may be taken.

Unnecessary Actions in Plans

The motivation for our work was that parallel plans may have many unnecessary actions. Table 1 shows the number of actions in parallel plans found by a satisfiability planner without restrictions on the number of actions and the smallest number of actions in any parallel plan of the same minimal length.[3] Note that the latter number in general is not the

[3] In few cases we were not able to establish the lower bound with certainty.

actions	number of steps 7	8	9
30	* 0.75	*142.73	-
31	0.04	0.22	0.09
33	0.06	0.13	0.18
35	0.06	0.26	0.37
37	0.06	0.08	0.32

Table 2: Comparison of runtimes for different plan lengths and numbers of actions for Logistics8_0

actions	number of steps 6	7	8	9
21	* 0.12	*42.2	-	-
22	* 0.15	1.85	13.07	40.92
23	* 0.23	0.44	2.81	22.66
24	* 0.43	0.06	1.63	8.45
25	0.04	0.16	1.27	1.09

Table 3: Comparison of runtimes for different plan lengths and numbers of actions for Driverlog8

minimum number of actions in any plan for that problem instance.

Impact of Parameters on Runtimes

We give the runtimes for evaluating formulae for different numbers of steps and actions for three easy problem instances in Tables 2, 3 and 4. Unsatisfiable formulae are marked with *. Evaluating formulae corresponding to a higher number of steps generally takes longer. Showing that a formula for given parameter values is unsatisfiable, that is, no plans of a given number of steps and actions exist, takes in general much more time than finding an satisfying assignment for one of the satisfiable formulae.

Comparison to Sequential Planning

We make a small comparison between the runtimes of parallel and sequential planning. With an encoding of cardinality constraints we can check whether a parallel plan with a given number of actions m and steps n exists. The computational cost of doing this can be compared to an encoding of sequential plans that restricts the number of actions and steps simultaneously to m.

Table 5 shows the results of the comparison. We determined the minimum number of steps needed in a parallel plan, and then measured the runtimes of finding a parallel plan with this number of steps and different numbers of actions as well as the runtimes of finding sequential plans with the same numbers of actions. The first column shows the name of the problem instance. The second column shows the number of steps in the parallel plans. The third column shows the number of actions allowed in the parallel and the sequential plans (and hence also the number of steps in the sequential plans.) The fourth column shows the SAT solver runtime for finding a satisfying assignment of the formula encoding sequential plans. The last column shows the SAT solver runtime for finding a satisfying assignment of the formula encoding parallel plans.

actions	number of steps 22	23	24
58	*10.50	*563.21	-
59	*14.43	*775.01	-
60	*19.03	*1064.88	-
61	*26.69	269.98	-
62	*39.66	109.70	326.04
63	*72.60	95.95	226.96
64	19.54	61.24	165.80
65	11.92	48.30	140.71
67	5.74	30.93	66.28
69	4.43	21.32	47.85
71	3.26	14.59	39.93
73	2.59	11.74	32.70
75	2.51	9.59	25.27
77	2.17	7.79	21.92

Table 4: Comparison of runtimes for different plan lengths and numbers of actions for Depots6

Overall we can summarize our observations as follows.

- It is more efficient to improve a parallel plan than a sequential one.
- The effectiveness of our techniques depends much on the type of the problem. On Logistics, Driverlog, Depots, Mystery and Satellite the plans could be greatly improved. For Philosophers or Optical Telegraph there was less improvement possible. For problem domains with no parallelism, like the sequential formalization of Blocksworld, parallel planning and the techniques presented in this work do not help of course.
- In most problems finding the plan with a given number of steps and the minimum number of actions (with Algorithm 1) yielded the optimal plan and it was not necessary to increase the number of steps (as with Algorithm 2).

Runtimes of Algorithm 2

We now illustrate the functioning of Algorithm 2. We first compute a parallel plan with the minimum number of steps and no constraints on the number of actions. Then we apply Algorithm 2 to find a plan with a smaller number of actions. In Table 6 the *total time* column shows the computation time (including initialization and previous steps) until a plan with the number of actions in column *actions* and the number of steps in column *steps* has been found. Unlike the results on previous tables these runtimes were not averaged over several runs because on different runs different sequences of plan lengths are encountered, skipping different plan lengths on different runs. For one of the four problems the algorithm was quickly able to prove the optimality of the last plan, in the other cases no better plans or optimality proofs could be found in several minutes.

problem	steps	actions	seq	par
Logistics4_0	6	26	0.11	0.01
		22	0.20	0.02
		20	0.41	0.01
Logistics5_0	6	29	129.09	0.02
		27	-	0.02
Logistics6_0	6	32	1.28	0.02
		28	43.26	0.01
		26	144.23	0.02
Logistics7_0	7	44	-	0.08
		40	-	0.08
Depots4	14	50	-	0.16
		40	-	0.21
		32	-	0.28
Depots5	18	71	-	0.39
		61	-	0.33
		56	-	0.87
Depots6	22	84	-	0.49
		70	-	0.50
Driverlog4	5	19	48.52	0.01
		18	46.11	0.01
Driverlog5	5	20	34.99	0.01
		18	53.48	0.01
Driverlog6	5	19	2.00	0.02
		14	1.32	0.02
		11	2.66	0.02
Driverlog7	5	20	76.58	0.03
		15	192.51	0.03
		13	123.01	0.03
Driverlog8	6	28	-	0.04
		25	-	0.04
Rover5	5	28	1.54	0.06
		22	114.61	0.13
Rover8	4	32	-	0.08
		26	-	0.08
Satellite5	4	28	45.10	0.06
		21	86.57	0.08
Satellite8	4	39	-	0.06
		31	-	0.81
Mprime4	6	35	10.36	7.81
		10	0.30	7.87
Mprime5	5	18	53.29	1.38

Table 5: Runtimes for evaluating formulae

problem	total time	steps	actions
Logistics8_0	0.08	7	41
	0.12	7	37
	0.19	7	34
	0.24	7	32
	0.30	7	31
	timeout		
Depot5	3.96	18	66
	4.38	18	63
	4.74	18	54
	6.06	18	53
	7.24	18	52
	12.68	18	51
	18.12	18	49
	timeout		
Mprime3	7.92	3	18
	8.45	3	4
	optimal		
Driverlog8	0.14	6	31
	0.18	6	25
	0.63	7	24
	1.12	7	23
	2.52	7	22
	timeout		

Table 6: Runtimes of Algorithm 2 on some problems

Comparison to Other Approaches

We can compare our framework to algorithms based on heuristic local search, for example the planners HSP and FF. A related comparison has been made in previous work (Rintanen 2004b). Two measures are of practical importance, the runtimes and the number of actions in the plans. For some problems like Logistics or Satellite HSP and FF very quickly find a good plan and therefore perform better than our satisfiability planner, no matter whether we have the plans improved or not. With other problems like Mystery or Depots, or more complicated problem instances in general, heuristic planners have great problems and often did not find any plan at all. In contrast, our satisfiability planner finds one plan relatively quickly and with a small further investment in CPU time improves the plan further. Overall we can say that as long as the heuristics work well planners based on heuristic local search seem to be a good choice. If this is not the case these planners either compute a bad plan with lots of actions after a lot of computation or are not able to find any plan at all in reasonable time.

Conclusions

We have investigated satisfiability in the framework of problem encodings that allow constraining both the number of steps and the number of actions in a plan. For this purpose we used encodings of cardinality constraints in the propositional logic and considered further techniques for reducing the sizes of these encodings further. Then we proposed planning algorithms that attempt to reduce the number of actions in a plan by repeatedly making the constraints on the number of actions stricter. Our experiments show that the algorithms can quickly improve the quality of plans found by a satisfiability planner, and that the approach is much more practical than the use of sequential plans.

Many of the benchmarks used in the experiments of this work have a rather simple structure and the unnecessary actions in the parallel plans may often be eliminated efficiently by a simple postprocessor that identifies irrelevant actions not needed for reaching the goals as well as pairs of actions of which the latter undoes the changes performed by the former. There are polynomial-time algorithms doing such simplifications (Lin 1998). For these benchmarks such postprocessors may be more efficient than the techniques we propose but for more challenging problems plan quality has to be addressed during plan search and cannot be postponed to a post-processing phase.

Some open questions remain. Rintanen (2004a) has considered parallelized algorithms for controlling a satisfiability planner in finding suboptimal plans and shown that parallelized strategies may dramatically improve the efficiency of planning. The results of our experiments in this paper suggest that after an initial plan has been found, improving the plan by successively changing the two parameters, number of steps and number of actions, would not benefit from a similar parallelized strategy. The question arises whether the use of cardinality constraints could speed up finding the initial plan – inside a parallelized algorithm that tests the satisfiability of formulae with several parameter values concurrently – by making some of the satisfiable formulae more constrained and therefore more efficient to solve.

References

Bailleux, O., and Boufkhad, Y. 2003. Efficient CNF encoding of Boolean cardinality constraints. In Rossi, F., ed., *Principles and Practice of Constraint Programming – CP 2003: 9th International Conference, CP 2003, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, 108–122. Springer-Verlag.

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

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

Dimopoulos, Y.; Nebel, B.; and Koehler, J. 1997. Encoding planning problems in nonmonotonic logic programs. In Steel, S., and Alami, R., eds., *Recent Advances in AI Planning. Fourth European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Computer Science, 169–181. Springer-Verlag.

Ernst, M.; Millstein, T.; and Weld, D. S. 1997. Automatic SAT-compilation of planning problems. In Pollack, M., ed., *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, 1169–1176. Morgan Kaufmann Publishers.

Garey, M. R., and Johnson, D. S. 1979. *Computers and Intractability*. San Francisco: W. H. Freeman and Company.

Haslum, P., and Geffner, H. 2000. Admissible heuristics for optimal planning. In Chien, S.; Kambhampati, S.; and Knoblock, C. A., eds., *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, 140–149. AAAI Press.

Hochbaum, D. S. 1998. Approximating clique and biclique problems. *Journal of Algorithms* 29:174–200.

Karp, R. 1972. Reducibility among combinatorial problems. In Miller, R., and Thatcher, J. W., eds., *Complexity of Computer Computations*, 85–103. Plenum Press.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *Proceedings of the 10th European Conference on Artificial Intelligence*, 359–363. John Wiley & Sons.

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, 1194–1201. AAAI Press.

Lin, F. 1998. On measuring plan quality (a preliminary report). In Cohn, A. G.; Schubert, L. K.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, 224–232. Morgan Kaufmann Publishers.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2004. Parallel encodings of classical planning as satisfiability. In Alferes, J. J., and Leite, J., eds., *Logics in Artificial Intelligence: 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004. Proceedings*, number 3229 in Lecture Notes in Computer Science, 307–319. Springer-Verlag.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2005. Planning as satisfiability: parallel plans and algorithms for plan search. Report 216, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Rintanen, J. 1998. A planning algorithm not based on directional search. In Cohn, A. G.; Schubert, L. K.; and Shapiro, S. C., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, 617–624. Morgan Kaufmann Publishers.

Rintanen, J. 2004a. Evaluation strategies for planning as satisfiability. In López de Mántaras, R., and Saitta, L., eds., *ECAI 2004: Proceedings of the 16th European Conference on Artificial Intelligence*, volume 110 of *Frontiers in Artificial Intelligence and Applications*, 682–687. IOS Press.

Rintanen, J. 2004b. Phase transitions in classical planning: an experimental study. In Dubois, D.; Welty, C. A.; and Williams, M.-A., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR 2004)*, 710–719. AAAI Press.

Valmari, A. 1996. Optimality results for median computation. In *Proceedings of IEEE Nordic Signal Processing Symposium (NORSIG'96)*, 279–282.