Engineering Efficient Planners with SAT

Jussi Rintanen
Griffith University and the Australian National University
Australia

Abstract. Planning with SAT has long been viewed as a main approach to AI planning. In comparison to other approaches, its high memory requirements have been considered to be a main obstacle to its scalability to large planning problems. Better implementation technology, especially addressing the memory use, together with a shift of understanding about SAT-based planning during the past ten years, enables planners that radically differ from those from the late 1990s. We discuss a SAT-based planning system that implements modern versions of virtually all components of first planners that used SAT, focusing on the new implementation technology for a compact clause representation that is both simpler and more effective than ones proposed earlier. Specifically, the decreased memory requirements enable the use of top-level solution strategies that lift the performance of SAT-based planning to the same level with other search methods.

1 INTRODUCTION

During the last decade, SAT, the prototypical NP-complete problem of testing the satisfiability of the formulas in the classical propositional logic [3], has emerged, due to dramatically improved SAT solvers, as a *practical* language for representing hard combinatorial search problems and solving them, in areas as diverse as model-checking [1], FPGA routing [25], test pattern generation [11], scheduling [4], haplotype inference [12], and diagnosis [23]. Planning as Satisfiability, which enjoyed a lot of attention in the late 1990s after the early works by Kautz and Selman [8, 9], remained less popular for more than ten years. This is somewhat surprising, considering the great successes of SAT in other areas of computer science. This situation can be traced back to properties of early SAT-based planning systems, which have made for example heuristic state-space search [2] more attractive for solving the planning community's standard benchmark problems. Only the most recent works have shown that SAT-based planning can achieve a performance comparable to the best implementations of other search methods [19, 18] for these problems.

In this work, we discuss the problems SAT based planning has been perceived to have and solutions that have been proposed. We show that what have been widely assumed to be *inherent* restrictions of the SAT approach to planning, are in fact not. Eliminating these issues leads to planning systems that are competitive with other planning systems with the standard planning benchmark problems. Specifically, we discuss two critical issues of SAT-based planning: potentially high memory requirements, and the necessity and utility of guaranteeing that plans have the shortest possible horizon length (parallel optimality). We will argue that these two issues are actually closely intertwined. We discuss implementation techniques that address these two issues, present one new implementation technique to reduce memory consumption, and show that the perceived disadvantages of SAT as a planning method have largely disappeared.

Our new technical contribution addresses the memory usage. Compact encodings of planning in SAT [21, 20] often reduce the memory requirements by a factor of 20 or more in comparison to Graphplan-based encodings [9, 10]. We will reduce the memory requirements further by a factor of 3 or more, not by changing the encoding, but its representation inside the SAT solver. Although the best encodings are quite compact, our experiments show that the additional reduction – used in the M and Mp planners we have presented earlier [19] – can still be crucial. We argue that our solution to compact clause representation is more effective than the one provided recently by Pralet and Verfaillie [14].

The structure of the paper is as follows. Section 2 describes the Planning and Satisfiability approach, the issues critical to its time and space complexity in practice, and makes a comparison between the state-of-the-art now and in the late 1990s. In Section 3 we present a new representation of binary clauses in SAT problems with recurring structure and its implementation in a modern SAT solver as well as a planner based on it. Section 4 presents a study of the impact of the compact representation on a planner's performance. Section 5 discusses related work before concluding the paper.

2 BACKGROUND

A classical planning problem is defined by a set F of facts (or state variables) the valuations of which correspond to states, one initial state, a set A of actions (that represent the different possibilities of changing the current state), and a goal description which expresses the possible goal states in terms of the facts F. A solution to the planning problem is a sequence of actions that transform the initial state step by step to one of the goal states. The classical planning problem can be translated into a SAT problem of the following form.

$$\Phi_T = I \wedge \mathcal{T}(0,1) \wedge \mathcal{T}(1,2) \wedge \cdots \wedge \mathcal{T}(T-1,T) \wedge G$$

Here I represents the unique initial state, expressed in terms of propositional variables $f@0$ where $f \in F$ is a fact, and G represents the goal states, expressed in terms of propositional variables $f@t, f \in F$. The formulas $\mathcal{T}(i, i+1)$ represent the possibilities of taking actions between time points i and $i+1$. These formulas are expressed in terms of propositional variables $f@i$ and $f@(i+1)$ for $f \in F$ and $a@i$ for actions $a \in A$.

The formula Φ_T is satisfiable if and only if a plan with $T+1$ time points exists. Planning therefore reduces to performing satisfiability tests for different T. The effectiveness of the planner based on this idea is determined by the following.

1. The form of the formulas $\mathcal{T}(i, i+1)$.
2. The way the values of t for Φ_T to be solved are chosen.
3. The way the SAT instances Φ_T are solved.

Below we will discuss the first two issues, and their interplay. The third, SAT solving, can be performed with generic SAT solvers [13] or SAT solvers specialized for planning [19].

2.1 Encodings of $\mathcal{T}(i, i+1)$

The encoding of transitions from i to $i+1$ as the formulas $\mathcal{T}(i, i+1)$ determines how effectively the satisfiability tests of the formulae Φ_T can be performed. The leading encodings are the factored encoding of Robinson et al. [21] and the $\exists$-step encoding of Rintanen et al. [20]. Both of them use the notion of *parallel* plans, which allow several actions at each time point and hence time horizons much shorter than the number of actions in a plan. The encoding by Robinson et al. is often more compact than that by Rintanen et al., but the latter allows more actions in parallel, which leads to shorter plans and smaller formulae. Both of these encodings are often more than an order of magnitude smaller than earlier ones such as the GraphPlan-based encodings of Kautz and Selman [9, 10] and their variants, still very recently widely used by other researchers, and also substantially more efficient [20, 22]. This is due to the very large quadratic representation of action exclusion in the Graphplan encodings. Rintanen et al. [20] and Sideris and Dimopoulos [22] show that eliminating logically redundant mutexes and improving the quadratic representation to linear dramatically reduce the size of the formulas.

An interesting property of Φ_T is that the transition relation formula $\mathcal{T}(i, i+1)$ is repeated for several time points i. The formula encodes the actions to change the state at time i to its successor at $i+1$. A priori, all possible actions are available at each time point. Attempts to take advantage of this have gone into two opposite directions. Kautz and Selman's GraphPlan-based encodings attempt to decrease the size of the SAT problems by eliminating some of the actions that are not possible in the initial state and other early states. In Section 3 we approach the repetitive structure of the formulas from a different angle.

2.2 Scheduling the solution of SAT instances

Kautz and Selman [9] proposed testing the satisfiability of Φ_T for different values of $t = 0, 1, 2, \ldots$ sequentially, until a satisfiable formula is found. This strategy, which is conceptually the same as breadth-first search, is asymptotically optimal if the k parameter corresponds to the plan quality measure to be minimized, as it would with sequential plan encodings that allow at most one action at a time. However, BlackBox uses Graphplan-style parallel plans for which the k parameter is meaningless because Graphplan's parallelism notion does not correspond to the actual physical possibility of taking actions is parallel. For STRIPS, Graphplan-style parallelism exactly matches the possibility of totally ordering the actions to a sequential plan [20]. Hence the parallelism can be viewed as a form of partial order reduction [6], the purpose of which is to avoid considering all $n!$ different ordering of n independent actions, as a way of reducing the state-space explosion problem. In this context the k parameter often only provides a weak lower bound on the sequential plan length. So if the minimality of k does not have a practical meaning, why minimize it? The proof that k is minimal is the most expensive part of a run of BlackBox and similar planners.

More complex algorithms for scheduling the SAT tests for different t have been proposed and shown both theoretically and in practice to lead to dramatically more efficient planning, often by several orders of magnitude [20]. These algorithms avoid the expensive proofs of minimality of the parallel plan length, and in practice still lead to plans of comparable quality to those with the minimal parallel length. These algorithms solve several SAT problems in parallel, for a bounded range of horizon lengths [20]. This is not feasible with the 1990s encodings of planning as SAT, as even solving a SAT problem for one horizon length was often impractical due to the size of the formulas. Hence the two issues, the size of the encodings and the strategies of considering different horizon lengths, are closely intertwined: without compact encodings and clause representations, efficient plan search, with multiple SAT instances solved simultaneously, is infeasible. The large size of the early encodings is probably the best explanation why the use of the sequential strategy has been popular still quite recently.

Figure 1 shows the performance improvements from BlackBox's successor SATPLAN06 with Graphplan-style plans ($\forall$-step plans [20]), parallel optimality and standard SAT solvers (SATPLAN06), to the same with a compact $\forall$-step encoding (A-opt), the same with more efficient $\exists$-step encoding (E-opt), the same without parallel optimality (E, corresponding to the M planner [19]), and finally, to a planning-specific heuristic (Mp [19]). The 998 STRIPS instances in the curves are from the planning competitions from 1998 until 2008.

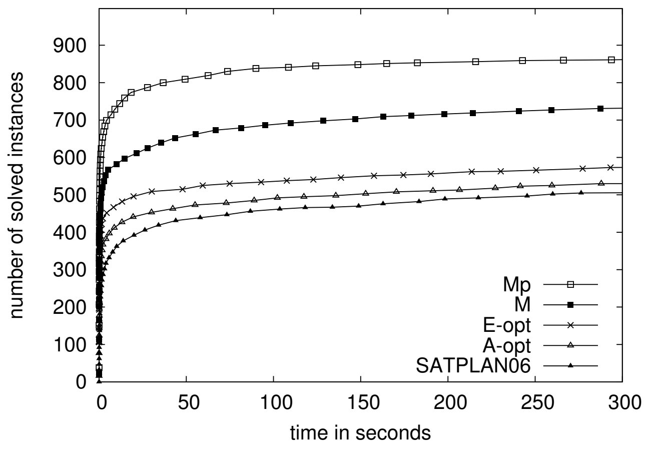

Figure 1. Progress in Planning with SAT

Of the above performance improvements, especially the one between E-opt and E is quite dramatic. The parallel optimality of earlier SAT-based planners (by having the planners perform a form of breadth-first search) leads to poor scalability and very high runtimes, due to the exponentially increasing runtimes of the unsatisfiable formulae. It is often very easy and sometimes almost trivial to find a plan given a sufficiently long horizon, but proving that no shorter plans exist is often very difficult.

Figure 2 depicts the gap between the longest horizon length with a completed unsatisfiability test and the horizon length for the found plan, for the Mp planner [19] and all planning competition instances solved by it. The dots concentrate in the area below 50 steps, but outside this area there are typically an area of 30 to 50 horizon lengths for which the SAT test was not completed, in the vast majority of cases because their difficulty well exceeded the capabilities of current SAT solvers. This explains why parallel strategies which avoid the unnecessary and expensive parallel optimality proofs are necessary for efficient planning.

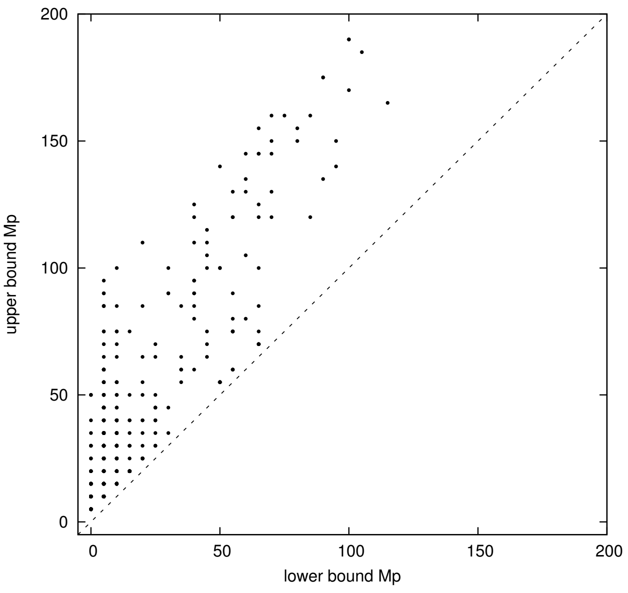

Figure 2. Lower and upper bounds of plan lengths

3 COMPACT REPRESENTATION OF BINARY CLAUSES

We next present a compact binary clause representation that substantially reduces the memory requirements of SAT solvers when they are solving problems such as planning, in which several copies of a large transition relation formula are needed. We have very successfully used this representation as a part of two planners, M and Mp [19], which scale up to very large planning problems much better than earlier SAT-based planners.

Instances $\mathcal{T}(0,1),\ldots,\mathcal{T}(T-1,T)$ of the transition relation differ only with respect to the time tags in the propositional variables. We propose a *schematic* representation for these clauses, which is instantiated with specific time tags whenever a specific instance of the clause is needed. We restrict this to binary clauses only, as they usually dominate the size of SAT encodings of planning. Concretely, for a clause $l_1@t_1 \vee l_2@t_2$ in $\mathcal{T}(i,i+1)$, with $\{t_1,t_2\} \subseteq \{i,i+1\}$, we represent the clause only once, parameterized with i. This reduces the number of occurrences of each binary clause in Φ_T from $T+1$ to 1. Further, our SAT solver can interleave the solution of Φ_T for several values of t (see Sections 2.2 and 3.3), and as the binary clauses are the same in every Φ_T, one copy of each clause suffices. When solving n instances of Φ_T for $T \in \{k,\ldots,k+n\}$, the total reduction in the number of occurrences of each binary clause is from $\sum_{i=0}^{n}(k+i) = k(n+1) + \frac{n(n+1)}{2}$ to 1.

3.1 Unit propagation with compact clauses

We use facts $f \in F$ in literals f and $\neg f$. We write $l@i$ when we mean $f@i$ for $l = f$ and $\neg f@i$ for $l = \neg f$. The complement $\bar{l}$ of a literal l is defined by $\overline{f} = \neg f$ and $\overline{\neg f} = f$.

Figure 3 gives the standard unit propagation procedure in which $\text{conseqs}(l) = \{l' | l \vee l' \in C\}$ represents all consequences of l inferred with binary clauses in the set C of input clauses (we implicitly assume commutativity: $l \vee l' \in C$ iff $l' \vee l \in C$.) When all the binary

```
1:  **procedure** propagate($l$,$v$);
2:  **if** $v \models \bar{l}$ **then return** false;
3:  $v := v$ modified so that $l$ is true;
4:  push $l$ into the stack;
5:  **repeat**
6:      pop $l$ from the stack;
7:      **for all** $l' \in \text{conseqs}(l)$ **do**
8:          **if** $v \models \overline{l'}$ **then return** false;
9:          **if** $v \not\models l'$ **then**
10:             $v := v$ modified so that $l'$ is true;
11:             push $l'$ into the stack;
12:         **end if**
13:     **end for**
14:     Propagate with $l$ and longer clauses;
15: **until** the stack is empty;
16: **return** true;
```

Figure 3. Unit Propagation

clauses come from the T copies of $\mathcal{T}(i,i+1)$, we can represent $\text{conseqs}(l)$ implicitly and more compactly. Propositional variables are represented as integers. There are V_1 propositional variables for each time point i, numbered from 0 to $V_1 - 1$. Now, a state or action variable x at time t, which we write as $x@t$, is represented as the propositional variable with index $x + tV_1$. Conversely, x in $x@t$ can be extracted with the modulo operation $x = x@t\%V_1$, and the time tag can be extracted as $t = x@t/V_1$. We define $\text{conseqs}(l@i)$ only for $i = 0$, and obtain its elements with binary resolution with the set C that consists of all the 2-literal clauses in $\mathcal{T}(0,1)$ and $\mathcal{T}(-1,0)$. We define $\text{conseqs}(l@0) = \{l'@t' | \overline{l@0} \vee l'@t' \in C\}$. Hence for all $l'@t' \in \text{conseqs}(l@0)$ we always have $t' \in \{-1,0,1\}$.

The unit propagation algorithm, extended to handle schematically represented two-literal clauses, is given in Figure 4. On line 7 we iterate over all consequences of the representative $l@0$ of $l@t$ at time point 0. These consequences are literals for the relative time points -1, 0 and 1. On line 8 we test whether the literal consequence was obtained with binary clauses in the actual clause set: we typically have literals $l'@-1 \in \text{conseqs}(l@0)$, and $l'@1 \in \text{conseqs}(l@0)$, and if the time tag t of the original literal $l@t$ was 0 or T, then some of these consequences would be for time points -1 or $T+1$, and was obtained with clauses not in the actual clause set.

3.2 Embedding in a SAT solver

This implementation of schematic binary clauses can embedded in any modern SAT solver implementation, requiring only minor modifications. The arithmetics for mappings between the timed and untimed representations through quotients and remainders causes a negligible overhead.

Notice that there are also binary clauses that cannot be, in general, schematically represented, including those representing disjunc-

```
 1: procedure propagate($l@t$,$v$);
 2: if $v \models \overline{l@t}$ then return false;
 3: $v$ := $v$ modified so that $l@t$ is true;
 4: push $l@t$ into the stack;
 5: repeat
 6:     pop $l@t$ from the stack;
 7:     for all $l'@t' \in$ conseqs($l@0$) do
 8:         if $t + t' \geq 0$ and $t + t' \leq T$ then
 9:             if $v \models \overline{l'}@(t + t')$ then return false;
10:             if $v \not\models l'@(t + t')$ then
11:                 $v$ := $v$ modified so that $l'@(t + t')$ is true;
12:                 push $l'@(t + t')$ into the stack;
13:             end if
14:         end if
15:     end for
16:     Propagate with $l@t$ and longer clauses;
17: until the stack is empty;
18: return true;
```

Figure 4. Unit Propagation for Schematic Binary Clauses

tive goals, and binary clauses generated by the conflict-driven clause learning (CDCL) procedure. Both these classes of binary clauses are represented in the conventional non-schematic way.

Representation of longer clauses and unit propagation for them remains unchanged. Binary clauses strongly dominate the SAT encoding size for most planning problems, and for STRIPS the only category of non-binary clauses is that of *frame axioms* which enumerate the possible causes for a change of a given state variable. The set of all frame axioms has a size that is linear in the size of the original (grounded) problem instance, and often represents only a very small fraction of all clauses in the SAT encoding.

3.3 Planner implementation

Since we cannot reduce an unbounded horizon length planning problem to SAT, nor do we want to search plans for one fixed-length horizon only, several SAT tests, for different horizon lengths, need to be performed. The main question here is how the CPU is allocated to solving the different SAT problems.

We have implemented a parallelized strategy for assigning CPU time to different horizon lengths, handled in connection with the *restart mechanism* of the SAT solver [7, 13]. The strategy B of Rintanen et al. [20] assigns CPU at a relative rate γ^t for horizon length t (not assigning any time to horizon lengths for which the computation has already been completed, of course), where $0 < \gamma < 1$, so that of any two consecutive horizon lengths t and $t + 1$, the latter gets γ times of the CPU of the former. The strategy is implemented with a counter c that is gradually increased, and when $c\gamma^t$ exceeds the number of runs (restarts) made for horizon length t so far, one more run will be made before restarting. The rate of increase of c is chosen so that the shortest currently active horizon length will be run on every round, and the longer horizon lengths less often. Instead of considering all horizon lengths, the planner looks at every third only, $0, 3, 6, 9, 12, \ldots$, which is better suited to the relatively long plan lengths encountered in practice. Accordingly, the planner only considers 20 SAT instances at a time, initially looking at horizon lengths $0, 3, 6, \ldots, 54, 57$.

Our SAT solver shares many of the data structures used in solving the SAT instances for several horizon lengths. For example, there is only one stack and only one clause database. The deletion of infrequently used long causes from the database takes place uniformly and simultaneously for all the SAT instances. The CDCL algorithm is run for all the instances in the same process and thread. The switching from one instance to another takes place during the frequent restarts of the CDCL algorithm. The switch operation is inexpensive. There is the obvious possibility to distribute the solution of the SAT instances to several threads, solved by different cores of the same CPU.

4 EXPERIMENTS

We illustrate the memory savings achieved by the compact clause representation with a brief comparison to a leading SAT solver Precosat (winner of the 2009 SAT solving competition and one of the best performing solvers for planning problems), and a more detailed comparison between alternative implementations of a planning-specific solver.

Consider the last instance (#20, ztravel-5-25b) of the ZenoTravel (2002) domain. Mp solves this problem in a couple of seconds with a peak memory consumption of 210 MB by the SAT solver when solving the SAT problems for horizon lengths 6, 9, 12, 15 and 18. In contrast, *Precosat* (one of the winners of the 2009 SAT competition) allocated $79+140+220+237+262 = 938$ MB for solving the same SAT problems for a run of the same duration (and without delivering a solution in that time.) This is almost 5 times the 210 MB used by our solver. For the last DriverLog instance 8-6-25 our memory consumption is 181 MB, and *Precosat* allocated $65 + 119 + 156 + 201 + 252$ = 793 MB (similarly without delivering a solution.)

Differences in Mp planner with and without the compact representation of binary clauses, respectively denoted by Mp and Mp-sparse, are similar, as shown in Figure 5. For two of the domains from the

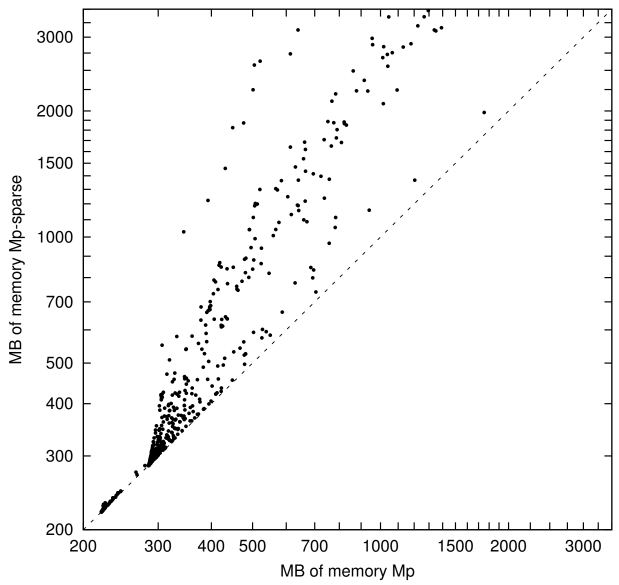

Figure 5. Comparison of memory consumption

planning competitions, Airport and Blocks World, the compact representation is critical: for half of the instances of both domains the planner runs out of memory (3 GB memory limit) when using a conventional binary clause representation. There is typically a difference of factor of 2 or 3. Sometimes the difference is small, most likely because the higher memory consumption of the non-compact representation is masked by the way the Linux GLIBC malloc works: allocation of small blocks of memory for the binary clauses reuses memory earlier allocated in small blocks by the front-end and later freed, while with the compact representation they remain unused.

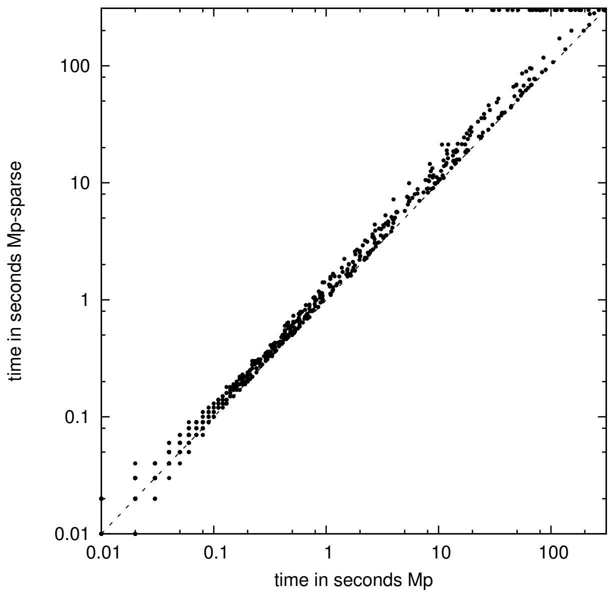

Figure 6. Comparison of runtimes by instance

Figure 6 shows the runtime differences between the two representations with 998 IPC instances (STRIPS). The compact representation is often faster, up to 40 per cent, because of fewer cache misses.

5 RELATED WORK

Pralet and Verfaillie [14] have used a compact representation of CSP problems that repeat the transition relation formula. However, unlike in our framework, Pralet and Verfaillie have a trade-off between compact encoding and efficiency: the compact slice encoding decreases the power of constraint solving, sometimes substantially so. In the worst case this leads to an exponential increase in the size of the search tree traversed. Our proposal, on the other hand, modifies only the implementation of unit propagation, does not decrease the power of SAT inferences, and improves efficiency on modern CPUs because of better cache utilization.

There is an earlier proposal for compact representation of binary clauses [17], based on compactly representing bicliques in the graph induced by binary clauses. This compacts the binary clauses at one time point, but not related binary clauses at different time points.

It is interesting to now have a new look at the issues faced by the SAT-based planners from late 1990s, more than 13 years later. Kautz and Selman [10] conclude from their comparison of BlackBox to the MEDIC planner [5] that "use of an intermediate graph representation appears to improve the quality of automatic SAT encodings of STRIPS problems". However, empirical or other demonstration of the utility of planning graphs as a basis of encodings has not later emerged. The useful information in planning graphs is the "persistent" fact mutexes, equivalent to *invariants* [16], which can be easily, and more efficiently, computed without constructing the planning graphs. The currently best encodings of planning in SAT encode the planning problem much more compactly than the planning graph encodings allow. The performance advantage observed by Kautz and Selman in the comparison to MEDIC was most likely due to the more explicit representation of some of the reachability information in planning graphs, which for current generation of SAT solvers [13] – radically different from the ones from 1990s – does not make a difference. Kautz and Selman do conclude: *"SAT encodings become problematically large in sequential domains with many operators, although refinements to the encoding scheme can delay the onset of the combinatorial explosion."* In retrospect, the "problematically large" encoding can be seen to have been caused by the quadratic encoding of action mutexes, which has later been reduced to linear [20], eliminating the problem. A further disadvantage of planning graphs is that they are incompatible with more efficient forms of parallel plans such as the $\exists$-step plans [20].

Performance of SAT-based planners has dramatically advanced in 13 years. One major change has been improved SAT algorithms. Further improvements have been in encodings and in the scheduling of the SAT tests. Consider Kautz and Selman's [10] Table 2, in which they list the solution times for rocket.$\{a, b\}$ and logistics.$\{a, b, c, d\}$ when finding minimal length Graphplan-style plans. Mp solves the same problems (including the proof of the parallel optimality) with speed-ups from 32 to 3300 over BlackBox. Without the parallel optimality proofs and by using encodings that allow more parallelism, all instances are solved in 0.02 seconds or less, with speed-ups of several thousands.

	rocket a b		logistics a b c d			
BlackBox	57	60	66	126	180	156
Mp par. opt.	0.02	0.02	0.02	0.24	5.19	4.88
Mp	0.01	0.01	0.02	0.01	0.01	0.02

6 CONCLUSIONS

We have presented new implementation technology for solving SAT problems and series of SAT problems that contain a sequence of instances of a transition relation formula, as found in SAT encodings of AI planning, bounded model-checking and other similar problems that require reasoning about state, action or transition sequences.

This new technology, together with other advances in SAT solving and encodings, has been implemented in a planning system, which has a performance comparable to the most scalable and efficient planners that use explicit state-space search, such as YAHSP [24] and LAMA [15]. A runtime comparison with all domains used in the planning competitions from 1998 until 2011 is given in Figure 7[1]. In addition to the M and Mp planners [19], we also include the runtimes of a new planner MpX which increases horizon lengths more aggressively than Mp, at the cost of a potentially higher memory consumption, but with substantially better scalability with problems with very long plans, as well as E-opt which is M producing makespan-optimal plans similarly to early SAT-based planners.

We argued that the high memory consumption of early SAT-based planners was a main obstacle to the use of parallelized planning

[1] YAHSP is not included in the diagram because of its restriction to STRIPS problems only.

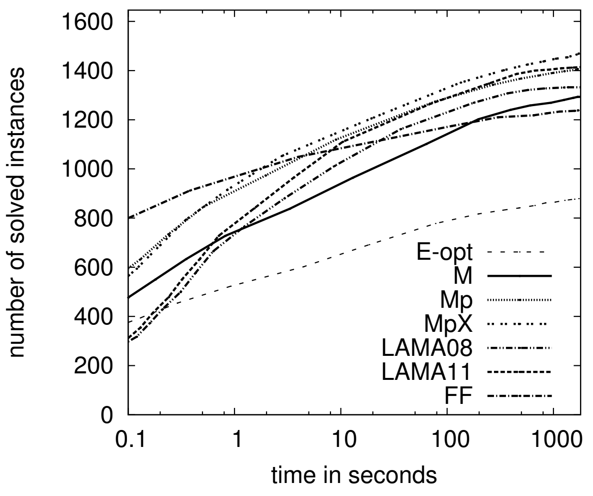

Figure 7. Number of instances solved for all domains in the planning competitions from 1998 until 2011. The best performing planners, MpX, LAMA11 and Mp, respectively have scores 41.10, 40.48 and 38.88, defined as sums of percentages of instances solved in 30 minutes for the 47 domains.

strategies, which in the past 10 years prevented SAT from being competitive in solving many of the standard benchmark problems. Reduction of memory consumption was essential for achieving truly efficient planners. This was achieved first through more compact problem encodings, with a linear rather than a quadratic number of clauses, and now in this work by more compact representation of clause sets inside a SAT solver. The more compact representation was shown to improve the performance of planning, first due to a reduction in memory overflows and premature termination of planner runs, and second, more systematically but less importantly, due to better cache utilization.

REFERENCES

[1] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu, 'Symbolic model checking without BDDs', in *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, ed., W. R. Cleaveland, volume 1579 of *Lecture Notes in Computer Science*, pp. 193–207. Springer-Verlag, (1999).

[2] Blai Bonet and Héctor Geffner, 'Planning as heuristic search', *Artificial Intelligence*, **129**(1-2), 5–33, (2001).

[3] S. A. Cook, 'The complexity of theorem proving procedures', in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151–158, (1971).

[4] Jamers. M. Crawford and Andrew B. Baker, 'Experimental results on the application of satisfiability algorithms to scheduling problems', in *Proceedings of the 10th National Conference on Artificial Intelligence*, pp. 1092–1097, (1994).

[5] Michael Ernst, Todd Millstein, and Daniel S. Weld, 'Automatic SAT-compilation of planning problems', in *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, ed., Martha Pollack, pp. 1169–1176. Morgan Kaufmann Publishers, (1997).

[6] P. Godefroid, 'Using partial orders to improve automatic verification methods', in *Proceedings of the 2nd International Conference on Computer-Aided Verification (CAV '90), Rutgers, New Jersey, 1990*, eds., Kim Guldstrand Larsen and Arne Skou, number 531 in Lecture Notes in Computer Science, pp. 176–185. Springer-Verlag, (1991).

[7] Carla P. Gomes, Bart Selman, and Henry Kautz, 'Boosting combinatorial search through randomization', in *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97) and 9th Innovative Applications of Artificial Intelligence Conference (IAAI-97)*, pp. 431–437. AAAI Press, (1998).

[8] Henry Kautz and Bart Selman, 'Planning as satisfiability', in *Proceedings of the 10th European Conference on Artificial Intelligence*, ed., Bernd Neumann, pp. 359–363. John Wiley & Sons, (1992).

[9] Henry Kautz and Bart Selman, 'Pushing the envelope: planning, propositional logic, and stochastic search', in *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pp. 1194–1201. AAAI Press, (1996).

[10] Henry Kautz and Bart Selman, 'Unifying SAT-based and graph-based planning', in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, ed., Thomas Dean, pp. 318–325. Morgan Kaufmann Publishers, (1999).

[11] T. Larrabee, 'Test pattern generation using Boolean satisfiability', *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **11**(1), 4–15, (1992).

[12] Inês Lynce and João Marques-Silva, 'Efficient haplotype inference with boolean satisfiability', in *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-2006)*, pp. 104–109. AAAI Press, (2006).

[13] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik, 'Chaff: engineering an efficient SAT solver', in *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, pp. 530–535. ACM Press, (2001).

[14] Cédric Pralet and Gérard Verfaillie, 'Slice encoding for constraint-based planning', in *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, CP 2009*, ed., I. P. Gent, volume 5732 of *Lecture Notes in Computer Science*, pp. 669–683. Springer-Verlag, (2009).

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

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

[17] Jussi Rintanen, 'Compact representation of sets of binary constraints', in *ECAI 2006. Proceedings of the 17th European Conference on Artificial Intelligence*, eds., Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, pp. 143–147. IOS Press, (2006).

[18] Jussi Rintanen, 'Planning with specialized SAT solvers', in *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11)*, pp. 1563–1566. AAAI Press, (2011).

[19] Jussi Rintanen, 'Heuristics for planning with SAT', in *Principles and Practice of Constraint Programming - CP 2010, 16th International Conference, CP 2010, St. Andrews, Scotland, September 2010, Proceedings.*, ed., David Cohen, number 6308 in Lecture Notes in Computer Science, pp. 414–428. Springer-Verlag, (2010).

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

[21] Nathan Robinson, Charles Gretton, Duc-Nghia Pham, and Abdul Sattar, 'SAT-based parallel planning using a split representation of actions', in *ICAPS 2009. Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, eds., Alfonso Gerevini, Adele Howe, Amedeo Cesta, and Ioannis Refanidis, pp. 281–288. AAAI Press, (2009).

[22] Andreas Sideris and Yannis Dimopoulos, 'Constraint propagation in propositional planning', in *ICAPS 2010. Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, pp. 153–160. AAAI Press, (2010).

[23] Alexander Smith, Andreas Veneris, Moayad Fahim Ali, and Anastasios Viglas, 'Fault diagnosis and logic debugging using Boolean satisfiability', *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **24**(10), (2005).

[24] Vincent Vidal, 'A lookahead strategy for heuristic search planning', in *ICAPS 2004. Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, eds., Shlomo Zilberstein, Jana Koehler, and Sven Koenig, pp. 150–160. AAAI Press, (2004).

[25] R. Glenn Wood and Rob A. Rutenbar, 'FPGA routing and routability estimation via Boolean satisfiability', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **6**(2), (1998).