Madagascar: Scalable Planning with SAT

Jussi Rintanen*
Department of Information and Computer Science
Aalto University
Helsinki, Finland

Abstract

The scalability of SAT to very large problems has been achieved in the last ten years, due to substantially improved encodings, SAT solvers, and algorithms for scheduling the runs of SAT solvers. This has lead to SAT-based planners that are dramatically different from the earliest implementations based on the late 1990ies technology. We discuss a SAT-based planning system that implements modernized versions of all components of earliest SAT-based planners.

Introduction

During the last decade, SAT, the prototypical NP-complete problem of testing the satisfiability of the formulas in the classical propositional logic (Cook 1971), has emerged, due to dramatically improved SAT solvers (Marques-Silva and Sakallah 1996; Moskewicz et al. 2001) as a *practical* language for representing hard combinatorial search problems and solving them, in areas as diverse as Model-Checking (Biere et al. 1999), FPGA routing (Wood and Rutenbar 1998), test pattern generation (Larrabee 1992), and diagnosis (Smith et al. 2005; Grastien et al. 2007).

Planning as Satisfiability, which enjoyed a lot of attention in the late 1990s after the works by Kautz and Selman (1996), has re-emerged as a strong approach to planning due to substantially improved problem encodings, SAT solvers, and search strategies. The main application is the classical planning problem (Rintanen 2012b), but the same ideas can be adapted to more complex forms of planning, or classical planning can be used as a subprocedure in algorithms for more general problems. These investigations have only started, with first breakthroughs obtained in temporal planning (Rankooh and Ghassem-Sani 2013).

These developments are not surprising, considering that the classical planning problem is equivalent to the simplest model-checking and reachability problems in Computer-Aided Verification, and that SAT and its extensions such as SAT modulo Theories (SMT) have had great successes in that area in the past ten years, including wide industry adoption.

*Also affiliated with Griffith University, Brisbane, Australia, and Helsinki Institute of Information Technology, Finland. This work was funded by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

In this paper, we will first give a brief description of the Planning and Satisfiability approach, discuss the issues critical to its time and space complexity in practice, and explain the factors that separate the state of the art now and ten years ago. 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).

The planning system Madagascar (also called M, Mp or MpC depending on its configuration) implements several of the innovations in planning with SAT, including compact and efficient encodings based on $\exists$-step plans (Rintanen, Heljanko, and Niemelä 2006), parallelized/interleaved search strategies (Rintanen 2004; Rintanen, Heljanko, and Niemelä 2006), powerful invariant algorithms (Rintanen 2008b), SAT heuristics specialized for planning (Rintanen 2010b; 2010a), and data structures supporting parallelized SAT solving with very large problem instances (Rintanen 2012a).

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 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 time points exists. Planning therefore can be reduced to

a sequence of satisfiability tests. 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 are chosen.
3. The way the SAT instances Φ_t are solved.

In the rest of the paper we will discuss each of these components of an efficient and scalable planning system that uses SAT.

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. (2009), and the $\exists$-step encoding of Rintanen et al. (2006). 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. Both of these encodings are often more than an order of magnitude smaller than earlier encodings such as those of Kautz and Selman (1996; 1999), and also substantially more efficient (Rintanen, Heljanko, and Niemelä 2006; Sideris and Dimopoulos 2010). This is due to the very large quadratic representation of action exclusion in early encodings. Rintanen et al. (2006) and Sideris and Dimopoulos (2010) show that eliminating logically redundant mutexes or improving the quadratic representation to linear dramatically reduces the size of the formulas.

The $\exists$-step plans allow more actions in parallel than the earlier most popular GraphPlan-style (Blum and Furst 1997) $\forall$-step plans (Dimopoulos, Nebel, and Koehler 1997; Rintanen, Heljanko, and Niemelä 2006). Further, the weaker conditions on parallelism for $\exists$-step plans often allow leaving out all constraints on the parallelity of actions, which further leads to smaller formulas than with $\forall$-step plans (Rintanen, Heljanko, and Niemelä 2006). Both factors, shorter horizon lengths and smaller encodings for action parallelism, substantially help improving the scalability of SAT-based planning.

In addition to constraints that are necessary for the correctness of planning, there are *redundant* constraints that logically follow from the necessary constraints, but that are still useful because they make such implicit facts explicit that would otherwise not be effectively inferred by the SAT solver (Rintanen 2008a).

Our planners use *invariants* (binary mutexes) to speed up SAT solving. Mutexes were first introduced in the GraphPlan planner (Blum and Furst 1997) for pruning a backward-chaining search, and they were soon noticed to be important also for SAT-based planning (Kautz and Selman 1996). The main reason for the utility of mutexes in planning is the representation of multi-valued state variables as sets of Boolean variables. That multi-valued state variables cannot be represented directly is a limitation of the PDDL language used by many planners.

We use a powerful algorithm for finding 2-literal invariants (Rintanen 2008b). The algorithm uses a fixpoint computation similarly to GraphPlan's planning graph construction, but works for a far more general input language that includes arbitrary disjunctions and conditional effects.

Scheduling the Solution of the SAT Instances

Kautz and Selman (1996) 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 is asymptotically optimal if the t 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, for the parallel $\exists$-step and $\forall$-step plans optimality of the t parameter is meaningless because the parallelism 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 (Rintanen, Heljanko, and Niemelä 2006). Hence the parallelism can be viewed as a form of partial order reduction (Godefroid 1991), 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 t parameter often only provides a weak lower bound on the sequential plan length. So if the minimality of t does not have a practical meaning, why minimize it? The proof that t is minimal was the most expensive part of the runs of early SAT-based 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 (Rintanen 2004; Zarpas 2004; Streeter and Smith 2007). 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. The most effective implementations of these algorithms solve several SAT problems (for different horizon lengths) in parallel.

Algorithm B (Rintanen 2004) runs an unlimited number of SAT solvers at varying rates, solving an sequence of SAT problems for formula $\Phi_0, \Phi_1, \Phi_2, \ldots$. Each SAT solver gets a fraction of the CPU that is proportional to γ^i, for some constant γ that satisfies $0 < \gamma \leq 1$ (we have very successfully used $\gamma = 0.9$). Hence each SAT test for Φ_i gets γ times the CPU the test for Φ_{i-1} gets. Most of the CPU is dedicated to short horizon lengths, but also longer horizon lengths get some CPU. In real-world implementations of the algorithm SAT solvers are started only for Φ_i for which an amount of CPU time is allocated that exceeds some positive threshold value. Our planners in their default configuration also limit the maximum number of SAT instances solved concurrently to 20, with new solvers started when earlier instances are found unsatisfiable.

Figure 1 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 and for all the instances considered by Rintanen (2010b). 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

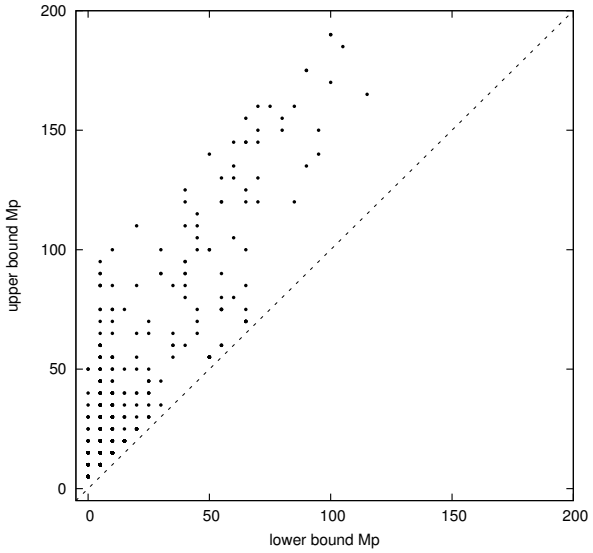

Figure 1: Lower and upper bounds of plan lengths

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 the use of the parallel strategies which avoid the expensive (but unnecessary) parallel optimality proofs are essential for efficient planning.

SAT Solving

Our planners are based on our own highly optimized SAT solver that implements the Conflict-Driven Clause Learning algorithm (Marques-Silva and Sakallah 1996; Moskewicz et al. 2001), together with many improvements more or less universally employed in best general-purpose SAT solvers, including phase-saving (Pipatsrisawat and Darwiche 2007), Luby-restarts (Huang 2007), and clause deletion based on literal blocking distance (Audemard and Simon 2009).

In addition to the standard VSIDS heuristic (Moskewicz et al. 2001), our SAT solver implements a planning-specific heuristic which in many cases fares much better than VSIDS on the standard benchmark sets (Rintanen 2012b). The heuristic simulates backward-chaining to identify relevant action variables to be used as decision variables, and leverages the current partial assignment maintained by the CDCL algorithm to focus on the most critical relevant actions. However, standard SAT solvers with VSIDS-style heuristics continue to be the strongest method for solving many combinatorially hard planning problems with relatively short horizon lengths (Porco, Machado, and Bonet 2011; Rintanen 2012b).

In addition to a planning-specific decision variable heuristic, our SAT solver employs a specialized representation for binary clauses (Rintanen 2012a) targeting planning and related state-space search applications such as model-checking. In all standard encodings of planning as SAT, the transition relation formula is replicated multiple times, with different time indices. Our SAT solver includes only one copy of all the binary clauses in the transition relation formula, and handles the varying time indices inside the unit propagation algorithm, with very low overhead. The representation often reduces the memory consumption of the planner to half or one third, and due to reduced cache misses also SAT solving runtimes are often reduced substantially (Rintanen 2012a).

planner	heuristic	scheduling strategy
M	VSIDS	B (geometric rates, linear horizons)
Mp	bwd	B (geometric rates, linear horizons)
MpC	bwd	C (constant rates, exponential horizons)

Table 1: Planner configurations

Versions of the Planner

There is no universal best configuration for our planner (similarly to any other planning method, or combinatorial search method in general), and we have introduced three major configurations which differ in terms of the heuristic and the SAT solver scheduling strategy. These configurations are listed in Table 1.

Planner *M* uses the standard VSIDS heuristic, limits search to plan lengths $5i$ for integers $i \geq 1$, and runs the SAT solvers at varying rates according to the geometric strategy B (Rintanen 2004; Rintanen, Heljanko, and Niemelä 2006). In addition to better encodings, the main difference to early planners that used SAT is the geometric B strategy, which can – in the worst case – be slower than the sequential strategy used by Kautz and Selman only by a small constant factor, but may be – and often in practice is – arbitrarily much faster.

Planner *Mp* is like *M* except that it replaces VSIDS with the heuristic based on backward-chaining which fares exceptionally well with standard planning benchmarks (Rintanen 2010b; 2012b), but often fares worse with smaller but combinatorially harder instances.

Planner *MpC* is like *Mp* but it replaces the horizon lengths $5i$ by horizon lengths $5(\sqrt{2})^i$, with all SAT solvers run at the same rate. *Mp* solves few problem instances with plans much longer than 200 steps due to the difficulty of proving inexistence of long plans and limits on the number horizon lengths considered simultaneously. *MpC* considers longer horizon lengths successfully, up to some thousands of steps, but beyond that it is severely limited by the availability of memory. It sometimes performs worse than *Mp* when the horizon lengths are short.

Conclusions

We have discussed a series of developments that have improved the efficiency and scalability of SAT-based planners dramatically since the early planners from the 1990ies and early 2000s, and that are all implemented in the Madagascar planner.

The single most important improvement – in terms of performance and scalability – was the adoption of parallelized

strategies that do not require the (unnecessary) proof of parallel optimality (Rintanen 2004; 2009). This improvement, together with compact encodings of parallel $\exists$-step plans (Rintanen, Heljanko, and Niemelä 2004), as implemented in the planner *M*, lifts the efficiency and scalability of SAT-based planning close to the level of the best modern planners that use other search paradigms, and clearly past planners prior to about 2004.

Further improvements for standard benchmark problems have been obtained by replacing general-purpose SAT-solver heuristics, such as VSIDS, by planning-specific ones which help focusing on actions that are relevant and which adapt to the current state of the SAT solving process (Rintanen 2012b), and with various smaller improvements. Finally, of course, a substantial difference to planning as SAT has been the steady and at times dramatic improvement of general-purpose SAT solving technology.

Existing techniques not used by our planners include the use of factored problem encodings for parallel plans (Robinson et al. 2009). With many problems these encodings have outperformed best non-factored encodings, but, as far as we know, the impact of factored encodings for example with parallelized SAT solving strategies (which are critical for high-performance planning) has not been investigated. Overall, the differences of encodings (which were the almost exclusive focus in research on planning with SAT for very long) have a smaller impact on the performance of a SAT-based planner than for example the SAT-solver scheduling strategies. Another method that is not currently used by our planners is approximate plan length upper bounds (Rintanen and Gretton 2013), which would help focusing the search on the horizon lengths most likely to yield plans quickly. The existing method sometimes yields practically significant upper bounds, based on SCCs of dependency graphs, but in many cases decomposition to SCCs is too coarse to be useful. The method is promising, but tighter bounds would be needed to have a substantial impact on planner performance and memory usage.

In summary, the current state of the art in Planning as SAT is characterized by developments in at least half a dozen different planner components. In many cases the improvements in the components have been orthogonal (for example, encodings, SAT solving algorithms, and scheduling of SAT solvers). Understanding dependencies between the different components, most notably between the encodings and algorithms for the SAT problem, would allow further progress in SAT-based methods for planning and state-space search problems in general.

References

Audemard, G., and Simon, L. 2009. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 399–404. Morgan Kaufmann Publishers.

Biere, A.; Cimatti, A.; Clarke, E. M.; and Zhu, Y. 1999. Symbolic model checking without BDDs. In Cleaveland, W. R., ed., *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, 193–207. Springer-Verlag.

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

Cook, S. A. 1971. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151–158.

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.

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

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, 305–310. AAAI Press.

Huang, J. 2007. The effect of restarts on the efficiency of clause learning. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2318–2323. AAAI Press.

Kautz, H., and Selman, B. 1996. 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*, 1194–1201. AAAI Press.

Kautz, H., and Selman, B. 1999. Unifying SAT-based and graph-based planning. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 318–325. Morgan Kaufmann Publishers.

Larrabee, T. 1992. Test pattern generation using Boolean satisfiability. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11(1):4–15.

Marques-Silva, J. P., and Sakallah, K. A. 1996. GRASP: A new search algorithm for satisfiability. In *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*, 220–227.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: engineering an efficient SAT solver. In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, 530–535. ACM Press.

Pipatsrisawat, K., and Darwiche, A. 2007. A lightweight component caching scheme for satisfiability solvers. In Marques-Silva, J., and Sakallah, K. A., eds., *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2007)*, volume 4501 of *Lecture Notes in Computer Science*, 294–299. Springer-Verlag.

Porco, A.; Machado, A.; and Bonet, B. 2011. Automatic polytime reductions of NP problems into a fragment of STRIPS. In *ICAPS 2011. Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling*, 178–185. AAAI Press.

Rankooh, M. F., and Ghassem-Sani, G. 2013. New encoding methods for SAT-based temporal planning. In *ICAPS 2013. Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, 73–81. AAAI Press.

Rintanen, J., and Gretton, C. O. 2013. Computing upper bounds on lengths of transition sequences. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2365–2372. AAAI Press.

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. 2006. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence* 170(12-13):1031–1080.

Rintanen, J. 2004. 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*, 682–687. IOS Press.

Rintanen, J. 2008a. Planning graphs and propositional clause-learning. In Brewka, G., and Doherty, P., eds., *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference (KR 2008)*, 535–543. AAAI Press.

Rintanen, J. 2008b. Regression for classical and nondeterministic planning. In Ghallab, M.; Spyropoulos, C. D.; and Fakotakis, N., eds., *ECAI 2008. Proceedings of the 18th European Conference on Artificial Intelligence*, 568–571. IOS Press.

Rintanen, J. 2009. Planning and SAT. In Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, number 185 in Frontiers in Artificial Intelligence and Applications. IOS Press. 483–504.

Rintanen, J. 2010a. Heuristic planning with SAT: beyond uninformed depth-first search. In Li, J., ed., *AI 2010 : Advances in Artificial Intelligence: 23rd Australasian Joint Conference on Artificial Intelligence, Adelaide, South Australia, December 7-10, 2010, Proceedings*, number 6464 in Lecture Notes in Computer Science, 415–424. Springer-Verlag.

Rintanen, J. 2010b. Heuristics for planning with SAT. In Cohen, D., ed., *Principles and Practice of Constraint Programming - CP 2010, 16th International Conference, CP 2010, St. Andrews, Scotland, September 2010, Proceedings.*, number 6308 in Lecture Notes in Computer Science, 414–428. Springer-Verlag.

Rintanen, J. 2012a. Engineering efficient planners with SAT. In *ECAI 2012. Proceedings of the 20th European Conference on Artificial Intelligence*, 684–689. IOS Press.

Rintanen, J. 2012b. Planning as satisfiability: heuristics. *Artificial Intelligence* 193:45–86.

Robinson, N.; Gretton, C.; Pham, D.-N.; and Sattar, A. 2009. SAT-based parallel planning using a split representation of actions. In Gerevini, A.; Howe, A.; Cesta, A.; and Refanidis, I., eds., *ICAPS 2009. Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, 281–288. AAAI Press.

Sideris, A., and Dimopoulos, Y. 2010. Constraint propagation in propositional planning. In *ICAPS 2010. Proceedings of the Twentieth International Conference on Automated Planning and Scheduling*, 153–160. AAAI Press.

Smith, A.; Veneris, A.; Fahim Ali, M.; and Viglas, A. 2005. Fault diagnosis and logic debugging using Boolean satisfiability. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(10).

Streeter, M., and Smith, S. F. 2007. Using decision procedures efficiently for optimization. In *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, 312–319. AAAI Press.

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

Zarpas, E. 2004. Simple yet efficient improvements of SAT based bounded model checking. In Hu, A. J., and Martin, A. K., eds., *Formal Methods in Computer-Aided Design: 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004. Proceedings*, number 3312 in Lecture Notes in Computer Science, 174–185. Springer-Verlag.