Systematic vs. Local Search for SAT

Holger H. Hoos[1] and Thomas Stützle[2]

[1] University of British Columbia, Computer Science Department,
2366 Main Mall, Vancouver, BC, V6T 1Z4 Canada
[2] Université Libre de Bruxelles, IRIDIA,
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium

Abstract. Due to its prominence in artificial intelligence and theoretical computer science, the propositional satisfiability problem (SAT) has received considerable attention in the past. Traditionally, this problem was attacked with systematic search algorithms, but more recently, local search methods were shown to be very effective for solving large and hard SAT instances. Especially in the light of recent, significant improvements in both approaches, it is not very well understood which type of algorithm performs best on a specific type of SAT instances.

In this article, we present the results of a comprehensive empirical study, comparing the performance of some of the best known stochastic local search and systematic search algorithms for SAT on a wide range of problem instances, including Random-3-SAT and SAT-encoded problems from different domains. We show that while for Random-3-SAT local search is clearly superior, more structured instances are often, but not always, more efficiently solved by systematic search algorithms. This suggests that considering the specific strengths and weaknesses of both approaches, hybrid algorithms or portfolio combinations might be most effective for solving SAT problems in practice.

1 Introduction

The satisfiability problem in propositional logic (SAT) is a central problem in logic, artificial intelligence, theoretical computer science, and many applications. Therefore, much effort has been spent on improving known solution methods and designing new techniques for its solution. This effort led to a continuously increasing ability to solve large SAT instances over the last years.

Traditionally, SAT instances are solved by systematic search algorithms of which the most efficient ones are recent variants of the Davis-Putnam (DP) procedure [5] like POSIT [7], TABLEAU [4], GRASP [22], SATZ [21], and REL_SAT [2]. These algorithms systematically examine the entire solution space defined by the given problem instance to prove that either a given formula is unsatisfiable or that it has a solution. Only recently, in the beginning of the 1990s, it was found that stochastic local search (SLS) algorithms can be efficiently applied to find solutions for hard, satisfiable SAT instances [27, 12]. SLS algorithms perform a biased random walk in the search space defined by all complete variable assignments, and test whether these are solutions in a non-systematic way. While SLS algorithms cannot prove unsatisfiability, in the past they have been found to outperform systematic SAT algorithms on hard subclasses of satisfiable SAT instances [1, 26].

Both techniques are largely different and comprehensive comparisons involving both types of algorithms are rather rare. In particular, it is not very well understood which type of algorithm should be applied to specific types of formulae. Only occasionally, systematic algorithms have been compared to local search based approaches [26] but most of these comparisons are

```
procedure DP(Φ, s)
    input SAT formula Φ, partial assignemt s, initially empty
    output satisfying assignment of Φ or "unsatisfiable"
    UnitPropagation(Φ, s);
    if Φ = ∅ then
        return s;
    if Φ contains empty clause then
        backtrack;
    l := chooseVariableToBranchOn(Φ, s);
    DP((Φ ∪ l, s ∪ {l := true});
    DP((Φ ∪ ¬l, s ∪ {l := false});
return "unsatisfiable";
end DP ;
```

Fig. 1. Outline of the Davis Putnam procedure for SAT; recent efficient variants differ mainly in the branching rule *chooseVariableToBranchOn*(Φ, s).

very limited in scope. Also, in the light of the recent significant improvements for both, systematic and local search methods, results from earlier comparisons might be outdated. To reduce this gap in our current knowledge we compare systematic and local search algorithms on a broad set of benchmark instances, covering hard Random-3-SAT instances, SAT-encoded, hard Graph Colouring instances, SAT-encoded planning problems from different domains, and instances from the DIMACS benchmark suite. To limit the computational burden of the study we focussed our investigation to the best performing systematic and local search algorithms currently available. In particular, from the available systematic search algorithms we tested SATZ and REL_SAT, from the SLS algorithms we used those based on the WalkSAT architecture [24].

The remainder of this article is structured as follows. In Section 2 we introduce the various algorithms used for our investigation and highlight their distinctive features. Next, in Section 3 we describe our experimental setup and present the results of our empirical comparative study. Finally, after discussing some related work in Section 4 we conclude with a summary of our main results and some suggestions for further research directions.

2 Algorithms for SAT

Systematic as well as local search algorithms for SAT are typically applied to formulae in conjunctive normal form (CNF). A CNF formula Φ over n truth variables $x_1, x_2, \ldots x_n$ (with domain $\{true, false\}$ each), is a conjunction of m clauses $c_1, c_2, \ldots, c_m$. Each clause c_i is a disjunction of one or more literals, where a literal l_j is a variable x_j or its negation $\neg x_j$, i.e. $c_i = \bigvee_{j=1}^{m_i} l_j$. A formula is satisfiable, if a assignment of truth values to all variables can be found which simultaneously satisfies all clauses; otherwise the formula is unsatisfiable.

2.1 Systematic Search Algorithms

The most efficient systematic search algorithms for SAT are based on the Davis-Putnam (DP) procedure [5] which implicitly enumerates all possible variable assignments. This is done by using a binary search tree in each node of which one variable is assigned a truth value, which is then fixed for the corresponding subtrees. The basic form of the Davis-Putnam procedure is

```
procedure LocalSearch(Φ, maxTries, maxSteps)
    input SAT formula Φ, maxTries, maxSteps
    output satisfying assignment of Φ or "no solution found"
    for i := 1 to maxTries do
        s := random truth assignment;
        for j := 1 to maxSteps do
            if s satisfies Φ then return s;
            else
                x := chooseVariable(s, Φ);
                s := s with truth value of x flipped;
            end if
        end for
    end for
return "no solution found";
end Local Search;
```

Fig. 2. Outline of a general local search procedure for SAT; actual SLS algorithms differ mainly in the variable selection function *chooseVariable*(s, Φ).

outlined in Figure 1. Starting with an empty variable assignment, in each recursive call of the algorithm the formula is first simplified by unit propagation, i.e., as long as a clause containing only one literal exists, the corresponding variable is assigned a value satisfying this clause and then deleted from the formula. If thus an empty clause is obtained, the current partial assignment cannot be extended to a satisfying one and backtracking is used to continue the search; if an empty formula is obtained, i.e., all clauses are satisfied, the algorithm returns a satisfying assignment. If neither of these situations occur, an unassigned variable is chosen and the procedure is called recursively after adding a unit clause containing this variable and its negation, respectively. If all branches are explored and no satisfying assignment has been found, the formula is found to be unsatisfiable. Systematic SAT algorithms are typically complete, i.e., they can decide the satisfiability (or unsatisfiability) of any given problem instance.

The effectiveness of the branching rule (procedure *chooseVariableToBranch*) has a very strong influence on the size of the search tree build [14] and is therefore crucial for the efficiency of the Davis-Putnam procedure. In particular for structured formulae further enhancements based on look-back techniques, like conflict directed backjumping and learning schemes, have led to improved DP variants [2, 22]. In this article we apply two of the most efficient currently known Davis-Putnam variants, SATZ [21] and REL_SAT [2]. SATZ strongly exploits heuristics geared towards maximising the efficiency of unit propagation in its branching rule. Additionally, it uses limited preprocessing of the input formula by adding resolvents of restricted length to the formula, for details we refer to [21]. REL_SAT uses look-back techniques and learning schemes to improve the performance and has been shown to be particularly efficient on structured instances (see [2] for more details).

2.2 Stochastic Local Search

Local search is a widely used, general approach for solving hard combinatorial search problems. Stochastic local search (SLS) can be interpreted as performing a biased random walk in a search space which, for SAT, is given by the set of all complete truth assignments. A general outline of a SLS algorithm for SAT is given in Figure 2. It starts with some randomly generated truth assignment and tries to reduce the number of violated clauses by iteratively flipping some variable's truth value. After a maximum of *maxSteps* such steps the algorithm restarts from a new random initial assignment. If after a given number *maxTries* of restarts no solution is found, the algorithm terminates unsuccessfully. SLS algorithms for SAT are typically incomplete, i.e., they cannot detect the unsatisfiability of a given problem instance.

SLS algorithms differ mainly in the heuristic for choosing the variable to be flipped in each search step (procedure *chooseVariable*) which is decisive for the final performance of the algorithm. In this article we focus on SLS algorithms based on the WalkSAT architecture [26, 24] which are among the best performing SLS algorithms for SAT currently known. These SLS algorithms use a two-stage variable selection process. In each step, first one of the clauses which are violated by the current assignment is randomly chosen. Then, according to some heuristic a variable occurring in this clause is flipped using a greedy bias to increase the total number of satisfied clauses. In this article we present computational results with five of the best performing WalkSAT variants. In WalkSAT with tabu-search [24] the strategy is to pick a variable that a minimises the number of breaks (the number of clauses which become unsatisfied by flipping a variable). The other strategies, Novelty, R-Novelty, Novelty$^+$, and R-Novelty$^+$ pick a variable that minimizes the number of unsatisfied clauses, additionally they use heuristics based on the idea that ties should be broken in favour of variables which have not been flipped for a longer time. Additionally, to avoid stagnation of the search, with a small, fixed probability instead of applying the usual variable selection heuristic, a variable is randomly chosen from the selected unsatisfied clause in Novelty$^+$, and R-Novelty$^+$. For details on these strategies, we refer to [24] and [18]. For all WalkSAT variants strategies, the so-called *noise-parameter* which controls the probability of flipping a variable not leading to the maximal increase in the number of satisfied clauses is of critical importance for the algorithm's performance. Note also that due to the stochastic choices inherent to these algorithms, the time for finding a solution to a given, satisfiable problem instance, is a random variable.

3 Experimental Comparison

Because of the inherent differences between systematic and local search algorithms conducting fair empirical comparisons is not straight-forward and involves some methodological problems. Two issues need to be addressed: which problem instances comparative study should be based on and how to measure algorithmic performance (or, equivalently, search cost) for each given problem instance.

3.1 Benchmark Problems

SLS algorithms are typically incomplete, which means that they cannot be used to prove the unsatisfiability of a given formula. Hence, comparisons between SLS algorithms and systematic search algorithms have to be restricted to satisfiable instances. Another possibility would be to run SLS algorithms to a given time limit and declare a formula as unsatisfiable if no solution is found. Obviously, this method is prone to false negatives since satisfiable formulae may be erroneously declared unsatisfiable. This has been suggested early in the development of SLS algorithms for SAT, but we are not aware of any empirical study following this approach. In the light of recent results in characterising SLS behaviour [16, 15, 18] and considering the fact that in time-critical application scenarios even theoretically complete algorithms will often become incomplete in practice if strict limits in computation time are enforced, this latter approach seems to be interesting; however, here we follow the more traditional approach in using satisfiable instances and assuming a scenario in which strict cutoffs (other than the ones imposed by the experimental environment) are not enforced and all algorithms typically run to completion.

To avoid empirical results which are overly biased by the type of problem instances used, a benchmark suite containing a broad variety of problem types has to be used. Additionally, we mainly focus on benchmark instances which are relatively hard for both types of search techniques. Finally, it is obviously desirable to use benchmark problems which have been widely used in the literature and are publically available. The benchmark suite we are using here comprises three different types of problems: test-sets sampled from Random-3-SAT, a well-known random problem distribution; test-sets obtained by encoding instances from a random distribution of hard Graph Colouring instances into SAT; and SAT-encoded instances from AI planning domains, in particular, from the Blocks World Planning domain and the Logistics domain [20]. All these benchmark instances are hard in general and difficult to solve for SLS algorithms. For the SAT-encoded problems, the hardness of the instances is inherent rather than just induced by the encoding scheme that was used for transforming them into SAT. In addition, we used some of the satisfiable benchmark instances from the second DIMACS challenge [19].

From this last category, we included only satisfiable instances. We excluded the `aim*.cnf` instances (generated by the AIM-generator) since they can be solved by polynomial simplification procedures and are therefore trivial for both approaches if this type of preprocessing is applied. From the `jnh*.cnf` instances only 16 out of 50 are satisfiable; these are randomly generated with variable clause lengths. Yet, we excluded these instances since initial experiments had shown they are easily solved by all algorithms studied here and we are already using a large set of hard Random-3-SAT. All benchmark instances used in this comparison are available through SATLIB, a benchmark collection of SAT instances, available on the WWW at the direction `http://www.informatik.tu-darmstadt.de/AI/SATLIB`.

3.2 Measuring Search Cost

While generally, for comparing algorithmic performance the use of machine and implementation independent operation counts is preferable over measuring CPU-time, here this is difficult because of the fundamental differences between the two classes of algorithms involved in our study. Whilst for comparisons between different systematic search algorithms, search cost is often measured as the number of explored nodes in the search tree (variable branches) and the number of unit propagations, the measure used for local search algorithms is typically the number of local search steps, i.e., the number of variable flips. In both cases, the CPU-time for these cost units typically depends on the problem size.

Fortunately, the algorithms investigated here are reasonably efficiently implemented, such that comparing CPU-times gives a realistic picture. Certainly, differences in the computation time may still be due to implementation details and could be affected by further optimisations in the implementations of either approach. Yet, such implementation-specific aspects may be negligible if the observed differences between the approaches are one or more orders of magnitude in computation time. Given the implementations used here, such differences, as we will see, can be observed and are most likely caused by properties of the algorithms. We conjecture that these can only be overcome by introducing substantially new algorithmic ideas.

For all our experiments, we also report the implementation-independent specific cost measures. For all WalkSAT algorithms, the operations counted are local search steps, i.e., variable flips. As an effect of the randomisation of the algorithm, the number of flips required for solving a problem instance varies widely between different runs. Therefore, to determine the search cost for a given problem instance, we run the algorithm at least 100 times with a cutoff parameter (maxSteps) setting which is high enough to guarantee a success rate close to 100%. From this data, we determine run-length distributions (RLDs) [16] from which the expected number of steps for solving the given instance can be easily estimated; this is used as a measure for search cost. For all WalkSAT variants, their performance critically depends on the setting of the noise parameter; to achieve close-to-peak performance, we therefore optimised this parameter for each problem size for the sets of randomly generated instances and sets of similar instances from the DIMACS set; for all other instances the noise parameter was optimised individually.

For SATZ, we chose the number of search steps as our primary measure for SATZ's search cost. This measure reflects the number of calls to a lower-level function which plays a critical role in variable selection and shows a strong correlation with CPU-time. Since SATZ is completely deterministic, the search cost per instance can be determined from a single run of the algorithm.

Finally, REL_SAT, like WalkSAT, as a consequence of its randomised decisions involved, e.g., in selecting the variables to branch on, has a large variability in run-time when repeatedly applied to the same problem instance. As a cost measure for REL_SAT, we used the number of variables labelled and obtained an expected number of these operations per solution as described above for WalkSAT. The same method was used for determining expected run-times (in CPU-seconds) per solution. Following the recommendations in [2], the learning order parameter was set to 3 for all experiments.

3.3 Results for Random-3-SAT

Uniform Random-3-SAT is a well-known family of SAT instance distributions which has been frequently used for empirically investigating the behaviour of SAT algorithms. Here, we use the test-sets as provided in SATLIB, which have also been used in [16, 15, 17]. Following the established procedures known from the literature [25], all instances are generated at the phase transition region [25, 4] and unsatisfiable instances are filtered out using systematic search algorithms.

In recent work, we identified R-Novelty$^+$ [15, 18] to be the best-performing stochastic local search algorithm for this subclass of SAT [15, 17]. From previously published results on the performance of systematic SAT algorithms, it can be concluded that SATZ is one of the best-performing systematic algorithms for this problem class; in particular, it solves hard Random-3-SAT instances more efficiently than REL_SAT [21]. Therefore, we empirically compared the performance of these algorithms sets of hard Random-3-SAT instances of varying size.

First, we analysed the distribution of search cost for R-Novelty$^+$ and SATZ across a test-set of 1000 Random-3-SAT instances with 100 variables and 430 clauses. As can be seen from Figure 3, except for a small number of instances (approx. 5%), R-Novelty$^+$ is more efficient than SATZ when neglecting the differences in CPU-time per search step between the two algorithms. In terms of absolute CPU-time, on a 400MHz Pentium II PC with 256M RAM under Linux R-Novelty$^+$ performs about 210,000 flips/CPU-sec, while SATZ executes ca. 83,000 search steps/CPU-sec. Thus, except for an extremely small fraction of the test-set, R-Novelty$^+$ outperforms SATZ. However, it should be noted that the variability in search cost across the test-set is significantly higher for R-Novelty$^+$ (stdev/mean $= 1.64$, ratio between the 0.9 and the 0.1 percentiles $q_{0.9}/q_{0.1} = 11.17$) than for SATZ (stdev/mean$= 0.54$, $q_{0.9}/q_{0.1} = 4.21$). The high variability and the heavy tail of the search cost distribution is typical for SLS performance on hard Random-3-SAT problems [15].

In [15, 17] it is shown that when comparing different SLS algorithms for SAT, the same instances tend to be hard for all algorithms. To investigate whether the same holds when comparing SLS algorithms and systematic SAT algorithms, we analysed the correlation between

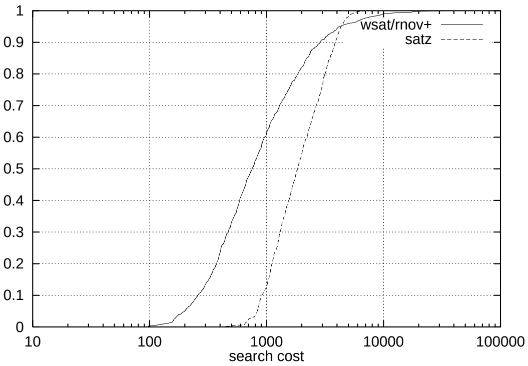

Fig. 3. Distribution of search cost across Random-3-SAT test-set of 1000 instances with 100 variables, 430 clauses each; search cost measured in expected number of flips per solution for R-Novelty$^+$, and number of search steps for SATZ.

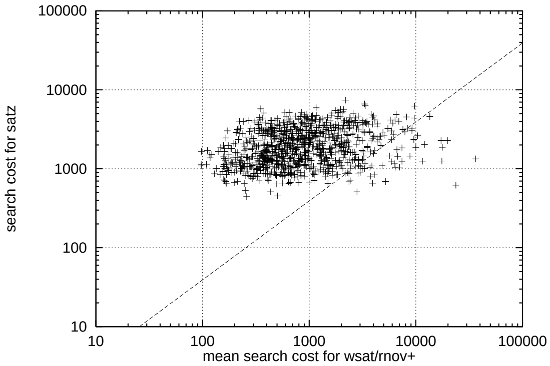

Fig. 4. Correlation between mean local search cost and systematic search cost across Random-3-SAT test-set of 1000 instances with 100 variables, 430 clauses each; search cost measured in expected number of flips per solution for R-Novelty$^+$, and number of search steps for SATZ. The line indicates points of equal CPU-time for the two algorithms.

search cost for both approaches across our test-set of 100 variable Random-3-SAT problems. Figure 4 shows the correlation data as a scatter plot, where each data point corresponds to one instance from the test-set. As can be easily seen from the plot, there is no strong correlation between the search cost for both algorithms. A correlation analysis confirms this result (correlation coefficient = 0.26); thus, the search cost for complete and local search seem to be only very slightly correlated. It can also be noted that for a major part of the test set, even without compensating for the different absolute CPU-time costs per search step, R-Novelty$^+$ is significantly more efficient that SATZ. The dashed line in the plot indicates points of equivalent CPU-time for both algorithms; thus, when comparing absolute CPU-time, for about 95% of the test-set, R-Novelty$^+$ is up to one order of magnitude faster than SATZ. When considering only those instances for which SATZ is more efficient than R-Novelty$^+$, again there seems to be no strong correlation between search cost for both algorithms.[3]

[3] There might be a tendency that the instances which are extremely hard for R-Novelty$^+$ are not particularly hard for SATZ; however, at this point, this observation cannot be considered to be statistically

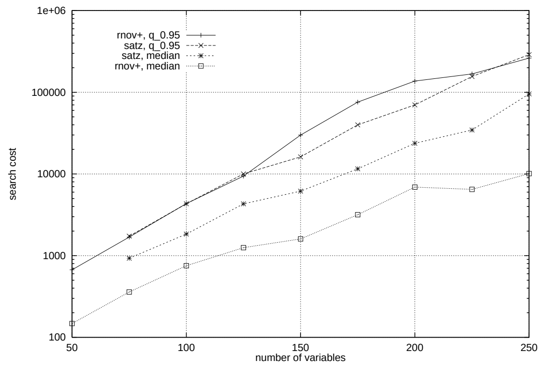

Fig. 5. Scaling of search cost with problem size for Random-3-SAT test-sets (≥ 100 instances each).

Next, we investigated the scaling of search cost with problem size for both algorithms on hard Random-3-SAT problems. For this investigation we generated a number of test-sets at the phase transition with 50 up to 250 variables (100 instances each, except the 50 and 100 variable instances, where the test-sets contain 1000 instances) and measured the distribution of search cost across each of these test-sets for R-Novelty$^+$ and SATZ. Figure 5 shows the dependence of the median and the 0.95 percentile on the problem size (number of variables) in a semi-logarithmic plot.[4] When neglecting the differences in CPU-time per search step between the algorithms, we observe that the median search cost for R-Novelty$^+$ is between 5 and 10 times lower than for SATZ; furthermore, this difference increases with problem size. However, when comparing the 0.95 percentiles, such differences cannot be observed. Nevertheless, when focusing on the CPU-time per single search step, we observe a difference in favour of R-Novelty$^+$, which increases with problem size (from ca. 2.6 R-Novelty$^+$ variable flips per SATZ search step for the $n = 100$ variable instances to ca. 3.2 for $n = 250$); thus, R-Novelty$^+$'s superiority over SATZ on Random-3-SAT is even more apparent when comparing CPU-times.

3.4 Results for Random Graph Colouring

The Graph Colouring problem (GCP) is a well-known combinatorial problem from graph theory: Given a graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices and $E \subseteq V \times V$ the set of edges connecting the vertices, find a colouring $C : V \mapsto \mathbb{N}$, such that neighbouring vertices always have different colours. We used Joe Culberson's random graph generator[5] for generating sets of 3-colourable flat random graphs with 50 to 200 vertices, with 100 instances each, where the connectivity (edges/vertex) was chosen such that the instances have maximal hardness (in average) for systematic graph colouring algorithms using the Brelaz heuristic [13]. These instances were then translated into SAT using a straight-forward encoding. These test-sets are available from SATLIB, where also a more detailed description can

significant.

[4] The data points for SATZ applied to the 50 variable test-set are omitted because for ca. 35% of these instances, SATZ takes longer than for any of the 75 or 100 variable instances to find a solution. This phenomenon is caused by the fact that for these formulae, SATZ adds a large number of resolvents to the original formula in a preprocessing step; it was not observed for any of the larger formulae.

[5] available from http://web.cs.ualberta.ca/~joe/Coloring/index.html, Joe Culbersons's Graph Colouring Page.

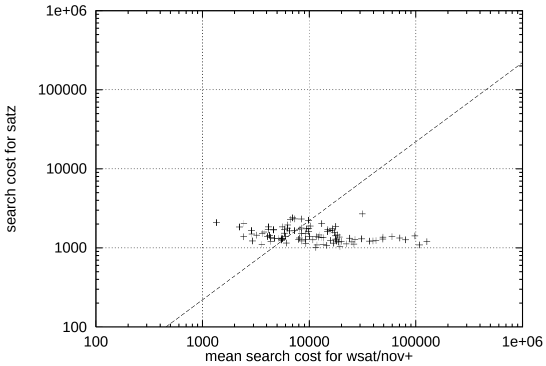

Fig. 6. Correlation between mean local search cost and systematic search cost across Flat Graph Colouring test-set of 100 SAT-encoded instances with 100 vertices, 239 edges; the line indicates points of equivalent CPU-time for the two algorithms.

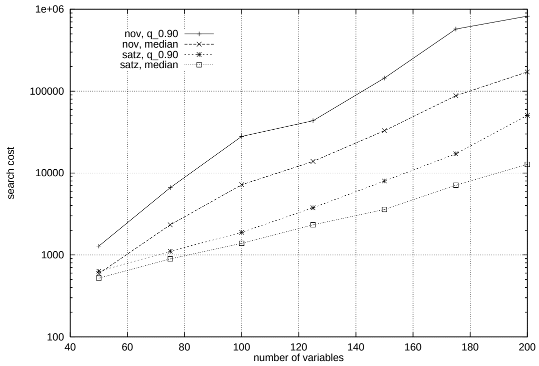

Fig. 7. Scaling of search cost with problem size for SAT-encoded Flat Graph Colouring test-sets (100 instances each).

be found. It has been shown in [15, 17] that, especially for the larger instances, Novelty is the best-performing SLS algorithm for this problem class. Nevertheless, for some instances it suffers from stagnation behaviour; as a consequence, here we use Novelty$^+$ [15, 18] which is less prone to this phenomenon.

The experiments were conducted analogously to the Random-3-SAT experiments described previously. As shown in Figure 6, there is a slight negative correlation between the search cost (correlation coefficient = -0.32). Furthermore it can be noted that the variance in search cost across the test-set is significantly lower for SATZ than for Novelty$^+$ (stdev/mean $= 0.22$, $q_{0.9}/q_{0.1} = 2.05$ for SATZ; stdev/mean$= 1.26$, $q_{0.9}/q_{0.1} = 10.15$ for Novelty$^+$). At the same time, when not considering differences in CPU-time per search step between the algorithms, the search cost for SATZ is for almost all instances significantly lower than for Novelty$^+$ (up to 2 orders of magnitude). When compensating for the differences in CPU-time per search step (ca. 4.54 Novelty$^+$ variable flips per SATZ search steps on our reference machine), the situation looks slightly different: in terms of CPU-time, Novelty$^+$ is more efficient than SATZ on a significant part of the test-set.

Figure 7 shows the results of a scaling analysis analogous to the one for Random-3-SAT.

				WalkSAT		SATZ		REL_SAT	
instance	#vars	#clauses	strategy	avg.flips	secs	#steps	secs	#lab. vars	secs
bw_large.a.cnf	459	4,675	Rnov+	6,053	< 0.1	354	< 0.1	1,765	< 0.1
bw_large.b.cnf	1,087	13,772	Tabu	152,104	1.62	544	< 0.1	8,818.5	0.48
bw_large.c.cnf	3,016	50,457	Tabu	$2.52 \cdot 10^6$	72.9	3,563	2.74	252,646	23.77
logistics.a.cnf	828	6,718	Rnov+	42,799.95	0.37	$6.4 \cdot 10^6$	10.45	30031.7	1.33
logistics.b.cnf	843	7,301	Rnov+	37,846.24	0.33	4672	0.22	268905	10.58
logistics.c.cnf	1,141	10,719	Rnov+	66,013.43	0.66	179679	1.76	$6.5 \cdot 10^6$	344.94
logistics.d.cnf	4,713	21,991	Nov+	121,391.08	1.96	$44.4 \cdot 10^6$	87.58	31550.4	1.28

Table 1. Comparison of solution times of WalkSAT, SATZ, and REL_SAT on instances of the blocks world planning and the logistics domain. #vars and #clauses give the number of variables and the number of clauses of each instance. The complexity measures (avg.flips, #steps, and #lab.vars) for the corresponding algorithms are explained in Section 2. The computation times are measured on a 300MHz Pentium II PC with 320M RAM under Linux.

Comparing search steps, SATZ shows a significantly lower search cost than Novelty$^+$ in the median and 0.95 percentiles for all problem sizes; this advantage increases with problem size. Here, all percentiles seem to show exponential scaling with problem size, but for SATZ the base of the exponential function characterising the growth is potentially smaller. Analogously to Random-3-SAT, our results also indicate that the variability in search cost increases with problem size for both algorithms, as can be seen when comparing the median and 0.95 percentile curves in Figure 7.

3.5 Planning Instances

As can be seen from Table 1, for the planning instance from the blocks world planning domain, both SATZ and REL_SAT show a significantly better performance than the best WalkSAT variant with SATZ being the best-performing algorithm. For the logistics domain, a different situation is encountered. The two instances `logistics.b` and `logistics.c` are solved by both, WalkSAT and SATZ, in short time, while SATZ takes significantly longer time on `logistics.a` and `logistics.d` than the WalkSAT algorithms or REL_SAT. Yet, REL_SAT performs worse than the other competitors on instances `logistics.b` and `logistics.c`. Hence, there is no clear dominance of one algorithm for this latter problem class but it should be noted that the best SLS algorithms generally seem to be quite competitive for all instances.

3.6 DIMACS Instances

Table 2 presents the results for some of the large graph colouring instances from the DIMACS benchmark set. These instances can be solved in relatively short time by the WalkSAT algorithms while SATZ and REL_SAT fail to find a solution for these instances within a time limit of 60 minutes. Hence, these SAT-encoded instances can only be solved in reasonable computation time[6] by using SLS algorithms.

Also on the `ii*` instances, originating from a SAT-encoding of problems in inductive inference, SLS algorithms are performing significantly better than the systematic algorithms. For

[6] All experiments on the DIMACS instances were run on a 300MHz Pentium II PC with 320M RAM under Linux.

			WalkSAT			SATZ		REL_SAT	
instance	#vars	#clauses	strategy	avg.flips	secs	#steps	secs	#lab. vars	secs
g125.18.cnf	2,250	70,163	Nov+	8,402.5	0.55	—	$> 60min$	—	$> 60min$
g125.17.cnf	2,125	66,272	Nov+	801,026.02	53.6	—	$> 60min$	—	$> 60min$
g250.15.cnf	3,750	233,965	Nov+	3,078.23	0.57	—	$> 60min$	—	$> 60min$
g250.29.cnf	7,250	454,622	Nov+	336607.71	86.8	—	$> 60min$	—	$> 60min$

Table 2. Comparison of solution times of WalkSAT, SATZ, and REL_SAT on instances of the DIMACS benchmark set.

			WalkSAT			SATZ		REL_SAT	
instance	#vars	#clauses	strategy	avg.flips	secs	#steps	secs	#lab. vars	secs
ssa7752-038.cnf	1,501	3,575	Rnov+	161,090.96	0.96	26083	0.15	1604.46	< 0.1
ssa7752-158.cnf	1,363	3,034	Rnov+	14,143.98	< 0.1	15259	0.1	1433.4	< 0.1
ssa7752-159.cnf	1,363	3,032	Rnov+	10,364.77	< 0.1	19015	0.12	1433.2	< 0.1
ssa7752-160.cnf	1,391	3,126	Rnov+	9,534.65	< 0.1	19273	0.12	1604.46	< 0.1

Table 3. Comparison of solution times of WalkSAT, SATZ, and REL_SAT on instances for circuit diagnosis.

example, WalkSAT solved all instances within at most 14000 variable flips on average, most of the instances (25 of 41) taking less than 1000 steps on average. Also the computing times for each of the instances were below 0.1 seconds. Yet, when compared with SLS algorithms, some of these instances are rather hard to solve for systematic algorithms. For example, SATZ did not solve one of the instances after 60 minutes, while REL_SAT could solve all instances, yet at the cost of rather long computation times, taking more than 800 seconds on average for the hardest instance. The `ssa*` instances, which originate from a test pattern program for checking "single-stuck-at" faults in VLSI circuits, are well solved by all of the competing algorithms (see Table 3). Only `ssa7752-038` is somewhat harder to solve for the SLS algorithm, while it is still easy for SATZ and REL_SAT.

In Table 4 a comparison between the two types of algorithms for the instances of learning the parity function are given. We report only the results for one of five instance of each size in the DIMACS set since the performance on the other instances is very similar. For these instances, SATZ and REL_SAT are clearly superior to the best SLS algorithms. Notice that all algorithms solve the small simplified instances `par8-*-c` very fast. When considering the unsimplified instances (`par8-*`), for the best local search algorithms (here R-Novelty and R-Novelty+ which perform equally well) the search cost increases by roughly a factor of 35, while for SATZ no significant difference in performance is observed. This is most probably caused by the fact that SATZ generally applies polynomial simplifications as a preprocessing step. These observations indicate that polynomial preprocessing of formulae can be very important when trying to solve instances, especially when using SLS algorithms. In this context it should also be noted that the planning instances used in Section 3.5 are generated using such simplification techniques.

Turning back to the comparison, the performance advantage of SATZ is more apparent for the larger instances (`par16-*` and `par16-*-c`). Here, the only SLS algorithm which

				WalkSAT		SATZ		REL_SAT	
instance	#vars	#clauses	strategy	avg.flips	secs	#steps	secs	#lab. vars	secs
par8-5-c.cnf	75	298	Rnov	4,052.12	< 0.1	298	< 0.1	273.56	< 0.1
par8-5.cnf	350	1,171	Rnov	133,591.30	0.51	393	< 0.1	1051.14	< 0.1
par16-5-c.cnf	341	1,360	Rnov	$4.12 \cdot 10^7$	173.4	39,5881	3.24	257,987	3.40
par16-5.cnf	1,015	3,358		—		216,976	2.04	$5.18 \cdot 10^6$	50.70

Table 4. Comparison of solution times of WalkSAT, SATZ, and REL_SAT on instances for learning the parity function.

were able to solve the simplified instances in reasonable time are R-Novelty and R-Novelty+. Yet, the average computing time for R-Novelty is already much higher than that of SATZ and REL_SAT. Possibly better results could be obtained by more parameter fine-tuning, but here systematic search algorithms like SATZ and REL_SAT appear to be the best choice. On the larger unsimplified instances we also observe an advantage of SATZ over REL_SAT, which we conjecture to be mainly caused by the SATZ's polynomial simplification preprocessing.

4 Related Work

Several comparisons between systematic and local search methods for SAT have been done in the past. The most extensive of these comparisons is probably that done at the DIMACS Challenge one Cliques, Coloring, and Satisfiability [19]. Yet, since then the performance of local and systematic search algorithms has increased strongly. Most other comparisons are rather limited in their scope. For example, in [26] GSAT and earlier WalkSAT variants are compared to a Davis-Putnam variant on the instances `ssa*` also used here. While their Davis-Putnam variant could not solve three of the four instances, SATZ and REL_SAT solve them very efficiently. This gives an indication of the significant progress in the development of systematic search algorithms for SAT and also documents the need for a systematic comparison of the more recent, best performing variants of both techniques. A comparison of the scaling of search cost for a Davis-Putnam variant and GSAT over a range of Random-3-SAT instances of varying constrainedness has been done in [11] and they found a better scaling behaviour of GSAT than for the systematic algorithm. Similarly, in [6] systematic and local search methods are compared for Graph-3-Colouring instances from both, the phase transition region and the easy regions. The comparison has shown that the systematic algorithm performed better at the phase transition region whereas for the easier instances the local search algorithms were found to be more efficient in finding solutions. More complete comparisons have been done exclusively among either systematic or local search algorithms. We refer to [21] and [15, 17] which are the most extensive comparisons known to us.

In the context of our results, combining local and systematic search methods appears to be attractive for efficiently and robustly solving SAT instances. One of the first such approaches for SAT was presented by Crawford [3]. It is based on first running a local search algorithm to determine clause weights which are then subsequently used to guide the branching heuristic of a systematic search algorithm. Another approach is taken by Mazure et.al. [23]. They use a combination of local search and systematic algorithms to prove unsatisfiability of large SAT formulae. This is done by first running local search to identify which clauses are most frequently unsatisfied. These clauses are then extracted from the formula and it is tried to prove unsatisfiability for this subset of clauses. While the viability of this approach was shown in [23], it is not clear whether this carries over to more recent algorithms, as for the formulae they tested the systematic algorithms applied in this article are much faster. Hence, it still has to be shown that by combining systematic and local search algorithms in one algorithm, the best "pure" methods can be outperformed.

5 Conclusions

In this paper we presented an empirical comparative study of some of the best currently known systematic and local search algorithms for SAT on a broad range of benchmark problems. Our results clearly indicate that currently, none of the two approaches dominates the other w.r.t. performance over the full benchmark suite. Instead, we found that for certain types of instances, local search algorithms are superior (like for hard Random-3-SAT or the large graph colouring instances from the DIMACS benchmark set) while for others (like hard graph colouring problems in flat graphs or the parity instances from the DIMACS benchmark set), systematic search seems to be significantly more efficient. Furthermore, we could show that for the randomised problem distributions like hard Random-3-SAT and hard graph colouring problems in flat graphs, there is only a very weak correlation (if any at all) between the search cost of both approaches across the instance distributions studied here.

Our study should be understood as an initial investigation, and as such, it leaves many interesting issues open for further research. Maybe the most important question to be answered is that of identifying the features of SAT instances which are responsible for the specific performance advantages of SLS or systematic search methods. Generally, there seems to be a tendency that SLS algorithms show superior performance for problems containing random structure (as in Random-3-SAT) and rather local constraints (as in the DIMACS Graph Colouring instances or the logistics planning domain), while systematic search might have advantages for structured instances with more global constraints. This view is supported by recent findings in studying a spectrum of graph colouring instances with varying degree of structural regularity [10].

The results presented here suggest that combining the advantages of systematic and local search methods is a promising approach. Such combinations could either be in the form of simple combinations, as briefly discussed in Section 4, of algorithm portfolios [9], or as truly hybrid algorithms. In this context, it would be interesting to extend our investigation to recently introduced randomised variants of systematic search methods, which show improved performance over pure systematic search under certain conditions [8]. Another issue which should be included in future research on SAT algorithms is the investigation of polynomial simplification strategies (like unit propagation, subsumption, restricted forms of resolution), which, when used as preprocessing steps, have been shown to be very effective in increasing the efficiency of SLS and systematic search methods in solving structured SAT instances [20].

References

1. M. Buro and H. Kleine-Büning. Report on a SAT Competition. Technical Report 110, Dept. of Mathematics and Informatics, University of Paderborn, Germany, 1992.
2. R.J. Bayardo Jr. and R.C. Schrag. Using CSP Look-back Techniques to Solve Real World SAT Instances. In *Proceedings of AAAI'97*, pages 203–208, 1997.
3. J.M. Crawford. Solving Satisfiability Problems Using a Combination of Systematic and Local Search. Paper presented at Second DIMACS Challenge: Cliques, Coloring, and Satisfiability, 1993.

4. J.M. Crawford and L.D. Auton. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence*, 81(1–2):31–57, 1996.
5. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proofing. *Communications of the ACM*, 5:394–397, 1962.
6. A. Davenport. A Comparison of Complete and Incomplete Algorithms in the Easy and Hard Regions. *Workshop on Studying and Solving Really Hard Problems* at CP'95, 1995.
7. J.W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.
8. C.P. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Trough Randomization. In *Proceedings of AAAI'98*, pages 431–437. MIT press, 1998.
9. C.P. Gomes and B. Selman. Algorithm Portfolio Design: Theory vs. Practice. In *UAI'97*, Morgan Kaufmann, 1997.
10. I.P. Gent, H.H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining Structure and Randomness. To appear in *Proceedings of AAAI-99*, 1999.
11. I.P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The Scaling of Search Cost. In *Proceedings of AAAI'97*. MIT Press, 1997.
12. J. Gu. Efficient Local Search for very Large-Scale Satisfiability Problems. *SIGART Bulletin*, 3:8–12, 1992.
13. T. Hogg. Refining the Phase Transition in Combinatorial Search. *Artificial Intelligence*, 81:127–154, 1996.
14. J.N. Hooker and V. Vinay Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.
15. H.H. Hoos. *Stochastic Local Search — Methods, Models, Applications*. PhD thesis, Department of Computer Science, Darmstadt University of Technology, 1998.
16. H.H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms — Pitfalls and Remedies. *Proceedings of UAI-98*, pages 238–245. Morgan Kaufmann Publishers, 1998.
17. H.H. Hoos and T. Stützle. Local Search Algorithms for SAT: An Empirical Evaluation. Submitted.
18. H.H. Hoos. On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. To appear in *Proceedings of AAAI-99*, 1999.
19. D.S. Johnson and M.A. Trick, editors. *Cliques, Coloring, and Satisfiability*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*. AMS, 1996.
20. H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of AAAI'96*, pages 1194–1201. MIT Press, 1996.
21. C.M. Li and Anbulagan Look-Ahead Versus Lock-Back for Satisfiability Problems. In *Proceedings of CP'97*, pages 341–355. Springer Verlag, 1997.
22. J.P. Marques-Silva and K.A. Sakallah, GRASP – A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM Int. Conf. on Computer-Aided Design*, 1996.
23. B. Mazure, L. Sais, and Éric Grégoire. Checking Several Forms of Consistency in Nonmonotonic Knowledge-Bases. In *Proceedings of FAPR'97*, pages 122–130, 1997.
24. D. McAllester, B. Selman, and H. Kautz. Evidence for Invariants in Local Search. In *Proceedings of AAAI'97*, pages 321–326, 1997.
25. D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings of AAAI'92*, pages 459–465, 1992.
26. B. Selman, Henry A. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In *Proceedings of AAAI'94*, pages 337–343. MIT press, 1994.
27. B. Selman, H. Levesque, and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of AAAI'92*, pages 440–446. MIT press, 1992.

This article was processed using the LaTeX macro package with LLNCS style