GSAT versus Simulated Annealing

A. Beringer[1] and **G. Aschemann** and **H. Hoos** and **A. Weiß**
TH Darmstadt, Alexanderstraße 10, 64283 Darmstadt, Germany
e-mail: antje@intellektik.informatik.th-darmstadt.de

Abstract. The question of satisfiability for a given propositional formula arises in many areas of AI. Especially finding a model for a satisfiable formula is very important though known to be $\mathcal{NP}$-complete. There exist complete algorithms for satisfiability testing like the Davis-Putnam-Algorithm, but they often do not construct a satisfying assignment for the formula, are not practically applicable for more than 400 or 500 variable problems, or in practice take too much time to find a solution. Recently, a (in practice) very fast, though incomplete, procedure, the model generating algorithm GSAT, has been introduced and several refined variants were created. Another method is Simulated Annealing (SA). Both approaches have already been compared with different results. We clarify these differences and do a more elaborate comparison showing that the performance of an already optimized variant of GSAT and an ordinary SA algorithm are more or less the same and that the attempts to further improve GSAT have lead to a procedure very similar to SA. Moreover we investigate how these algorithms can be parallelized.

1 Introduction

In the last two years, the very fast, though incomplete, model generating algorithm GSAT (first described in [16]) for propositional satisfiability problems attracted many researchers because of its simplicity. They investigated and tried to further improve GSAT during the last two years. Another, nearly 30 years old, method is Simulated Annealing (SA) [9], which is often used for optimization and constraint satisfaction problems and can be easily adapted to neural networks. Consequently, it is easy to parallelize. Both approaches have already been compared, with different results [15, 17]. At first sight, both comparisons seemed unfair in some points, so we tried to clarify these differences and do a more elaborate comparison of the amount of work done by the two algorithms. We run GSAT and SA on *identical* problems, using the *same* random number generator for the starting points of both algorithms. We considered propositional random formulas in conjunctive normal form, each clause consisting of three literals (3-CNF), comparing problems with 100, 200, and 300 variables and a ratio of 4.3 for clauses/literals. This has been reported to be a class of very hard problems for the Davis-Putnam-Procedure as they are neither under- nor over-constrained.
We found that the performance of an already optimized variant of GSAT and an ordinary SA algorithm are more or less the same and that the attempts to further improve the performance of basic GSAT lead to an algorithm very similar to SA. As it is expected, that in the future the most powerful computers will be massively parallel ones, we also investigated how GSAT and SA can be parallelized. It seems that SA is better suited for parallelization than GSAT as it does not use global information to make its decisions. This is explained in detail in Section 5. In Section 2 we give an overview over the similarities and differences of both approaches. Experimental results are reported and evaluated in Section 3 and related work is presented in Section 4.

2 GSAT and Simulated Annealing

GSAT and SA mainly rely on the same ideas. Both operate on a kind of *energy function* E, searching for its global minima. Propositional formulas in conjunctive normal form (CNF) can be related to such a function in the following way. We construct the energy function such that each of its variables corresponds to one of the propositional variables of the formula and for a given 0-1-assignment of these variables the value of the energy function gives the number of unsatisfied clauses in the propositional formula. For example the formula $(A \vee B) \wedge (\neg B \vee C)$ is expressed by the energy function $1 - A + AB - BC$. Its models correspond to the variable assignments with least value of the energy function. Consequently, the energy function is positive semi-definite and has global minima with value 0 if the formula is satisfiable, corresponding to the truth assignments satisfying all clauses of the propositional formula. I.e., the global minima of the energy function give the models of a satisfiable formula. As GSAT and SA search for global minima of a given energy function, they can be used to find models of a given satisfiable formula. Unfortunately, both may get trapped in *local* minima which are not global. So strategies to escape from these local minima are necessary. We will first describe the general idea underlying both algorithms and then go into the details. GSAT and SA both start with a random truth assignment for the variables. If this is not already a satisfying assignment for the underlying propositional formula, then the procedure *hillClimb*
successively selects variables, whose assignment will be alternated (a so-called *flip* of the variable). This selection is done according to certain preferences depending on the energy change, the flip of a given variable V will cause. This difference $\Delta E(V)$ is called the *score* of the variable. A very low (negative) score means a deep decrease of the energy and

[1] A. Beringer is supported by DFG, project MPS, grant no. HO 1294/3-2.

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

usually is preferred. The selection of variables is repeated until a global minimum of the energy function E is found or the maximal number of iterations is reached. The search for a satisfying assignment is restarted up to MAXTRIES times with a new initial random assignment. This gives a possibility to escape from local minima.

A unifying algorithm *GenSAT* for both methods within the GenSAT-framework introduced by [5] is given in Figure 1. E denotes the energy function corresponding to the input formula (containing N variables), MAXCYCLES and MAXTRIES are natural number parameters. *hillClimb* computes new assignments, *pick* picks one variable randomly, and *flip* performs a flip. *rnd* generates a random number lying in the given interval and *score* computes the energy change resulting from a certain flip in the current assignment. *prob* and *temperature* are described in detail in subsection 3.1. The usual notation for GSAT and SA algorithms can be found in [16] and [17, 2], respectively.

```
procedure GenSAT(E)
  for i := 1 to MAXTRIES do
    A := random assignment for variables in E;
    if E(A) = 0 then return A
    else
      for j := 1 to MAXCYCLES do
        A := hillClimb(E, A, j);
        if E(A) = 0 then return A
      end for;
  end for;
  return "no satisfying assignment found";
end GenSAT;

procedure hillClimb(E, A, j) % for GSAT
  for V := 1 to N do
    ΔE(V) := score(A, V);
  end for;
  if rnd(0,1) ≥ 0.5 then
    possflips:={V | ΔE(V) is minimal}
  else
    possflips:={V | V ∈ unsatisfied clause};
  V := pick(possflips);
  A:= flip(A, V);
  return A;
end hillClimb;

procedure hillClimb(E, A, j) % for SA
  T := temperature(j);
  for V := 1 to N do
    if rnd(0,1) ≥ 0.5 then
      ΔE(V) := score(A, V);
      accept:= prob(ΔE(V), T)
    else accept:= false;
    if accept then
      A:= flip(A, V);
      if E(A) = 0 then return A;
  end for;
  return A;
end hillClimb;
```

Figure 1. A general satisfiability algorithm for GSAT and SA and *hillClimb* for our GSAT and SA variants.

The procedure *hillClimb* realizing GSAT first updates the scores of all variables and then computes a list `possflips` of variables that may be flipped. One of these variables is chosen and the new assignment A resulting from its flip is returned. If this assignment does not already evaluate to a global minimum of the energy function E, *hillClimb* is restarted up to MAXCYCLES times. As in each execution of *hillClimb* exactly one variable is flipped, MAXCYCLES corresponds to the parameter MAXFLIPS in [15, 16]. The given *hillClimb* variant already includes an improvement over basic GSAT called *random walk*, first introduced in [15]: In basic GSAT, `possflips` always contains the variables with the least score (which will be negative in general), i.e. the variables, whose flip results in the deepest decrease of the energy (that is why GSAT is a *greedy* algorithm). If random walk is included, then with probability p (here $p = 0.5$), this list contains all variables occurring in an unsatisfied clause. Consequently, it includes all variables with a negative score, especially those with the least negative score. But not every variable occurring in an unsatisfied clause has a negative score. So the variable selected from this list may cause an *upwards* move on the energy surface. This is a further method to escape from local minima, as it allows to "jump" over shallow hills in the energy landscape. We used this variant of GSAT as it performs significantly better than the basic version [15].

For SA, MAXCYCLES is the number of iterations through all variables of the formula. Also the procedure *hillClimb* behaves different. It implements a feature inherent to SA which in a different manner than random walk also gives the possibility to make random *upwards* moves on the energy surface. Here the `possflips` list of variables which may be flipped need not be computed as the variable selection is purely local. During one iteration through the variables more than one flip may occur. In our SA variant, each variable is updated with probability 0.5. It then is determined by means of a probability distribution $prob(\Delta E(V), T)$ depending on the variable V's score $\Delta E(V)$ and the current *temperature* T,[2] if a selected variable is actually flipped. Initially, this temperature is set to a rather high value MAXTEMP and then decreases according to an *annealing schedule* which may be arbitrarily chosen. SA is theoretically guaranteed to find a global minimum in the limit if the annealing is done in infinitesimally small steps until the temperature really reaches 0 (the distribution then causes gradient descent to be performed) [7]. But this result is not suitable in practice, because the convergence time tends to infinity as the difference between two values of the temperature tends to zero, and we consequently must approximate the infinitesimally slow annealing. So, SA in practice is also incomplete (like GSAT). The annealing schedule usually depends on the number of iterations j through the algorithm and a certain maximal difference ΔT between two values of the temperature. It is necessary to find a compromise between very slow annealing (which improves the probability that a global minimum will be found), and keeping the time for the search short. We tested several annealing schedules including geometric ones, those with constant annealing steps and even with constant temperature (i.e. no annealing) and found that

[2] This terminology stems from the analogy to physical systems.

the following schedule performed best[3]:

$temperature(1) = \text{MAXTEMP},$
$temperature(j+1) = temperature(j) - \Delta T/j, \quad (j \geq 1).$

In our experiments, we additionally used a minimal temperature MINTEMP and stopped the annealing if the temperature reached this value. Beyond this point, gradient descent was performed until the nearest local minimum was reached (which is equivalent to setting the temperature to 0). The distribution for the choice of the flipping variable was a combination of the so-called Metropolis distribution [10] and gradient descent, if MINTEMP or MAXCYCLES was reached. We also tested the performance of SA with the Glauber distribution [6], where even the probability for *improving* flips depends on the score (i.e. a more greedy choice of the variables), but we found the performance of Metropolis to be superior.

$$P(prob(\Delta E, T) = \text{true}) = \begin{cases} 1 & \Delta E < 0 \\ \exp(-\frac{\Delta E}{T}) & \Delta E \geq 0 \;\& \\ & T \geq \text{MINTEMP} \;\& \\ & j \leq \text{MAXCYCLES} \\ 0 & \text{otherwise} \end{cases}$$

This probability distribution means, that variables with negative score, i.e. whose flip will *decrease* the energy, are *always* flipped if they are selected, whereas variables increasing the energy are flipped only with a probability depending on the amount of increase they cause (or never in the third case). This kind of choice is indifferent towards the extent of improvement a flip causes, in contrast to GSAT which greedily chooses always a variable with least score, if no random walk is performed. Above all, our SA algorithm also contains two methods to escape from local minima: random upwards moves, which is a feature inherent to all SA algorithms, and restarting the search after a given time.

Up to this point we described SA independent from the underlying data structure. But energy functions easily can be related to symmetric neural networks. Each energy function uniquely describes a neural network, whose binary threshold units correspond to the variables of the energy function, such that the network's stable states correspond exactly to the local minima of that function [8, 12]. We therefore can use a neural network for SA and the search of models of satisfiable propositional formulas [11]. If the network is in a stable state and its energy has a global minimum, then the activation of its units gives a model for the underlying propositional formula. This is an interesting feature as it leads to easy parallelizability of the algorithm and automatically includes several algorithmic improvements also introduced to GSAT, as described in the following sections.

3 Experimental Results

In our experiments we used random propositional formulas in 3-CNF with a ratio of 4.3 for clauses/variables. This has been reported to be a characterization of very hard formulas, as they are neither under- nor over-constraint [3]. We tested problems with 100, 200, and 300 variables and set the parameters as follows: $\Delta T = 0.01$, MAXTEMP $= 0.3$, MINTEMP $= 0.01$, MAXTRIES $= 10$. We let GSAT and SA do all 10 tries, counting how many assignments they found in average. This way of comparison avoids too much influence of a certain initial assignment. By averaging the amount of work to find an assignment over 10 tries, we lower the possibility that an actually very difficult problem (even the problems with ratio 4.3 seem to vary very much in their difficulty) is solved very quickly only because the initial assignment was near to the solution, and vice versa for the easier problems.

We set MAXCYCLES $= 1000$ for SA and computed an appropriate setting for GSAT determining the maximal number of flips allowed. In one cycle of SA, about half of the variables are updated ($N/2$). By doing the experiments we found an almost constant ratio of updates per flip of about 5.5. This means, that per cycle about $N/(2*5.5) = N/11$ flips are performed. Allowing 1000 cycles then means that up to $1000 * N/11 \approx 100 * N$ flips are possible. We therefore used a varying MAXCYCLES parameter of $100 * N$ for GSAT. We ran both algorithms on 100 instances of the 100 and 200 variable formulas and 50 instances of the 300 variable problem. The formulas were generated with the random formula generator obtained from Bart Selman, and they were constructed such that they were known to be satisfiable. One should note that this may lead to easier problem instances than filtering out unsatisfiable formulas after generation[4] But nevertheless the performance of both algorithms remains comparable. This just explains why our results for GSAT are much better than those reported in [15].

For the SA algorithm we used the neural network simulator ICSIM written in the object oriented language SATHER which was developed at ICSI [13]. We simulated a Boltzmann machine with higher order connections [12]. The units in such a network correspond directly to the variables in the propositional formula, which also determines the type of connections between the units. The minima of the energy function correspond to the stable states of the network [8, 12]. The statistical results of the experiments are shown in the following table, where we compared GSAT and SA in the number of flips (F), updates (U) and assignments (A) found. The ratio U/F for GSAT was assumed to be 25.8 in average as explained in subsection 3.1.

	GSAT, random walk			SA, Metropolis		
N	Ass.	F/A	U/A	Ass.	F/A	U/A
100	100%	520	13416	99.6%	642	3537
200	99%	1622	41848	99.1%	2547	14110
300	99.8%	2394	61765	99.6%	4889	26743

3.1 How to interprete these results?

When counting the number of updates and flips performed per assignment, averaged over 100 problems, we found a much lower ratio of updates/flips for SA than the one reported for GSAT ([17]). This becomes clear if one considers the differences in the algorithms for GSAT and SA: Whereas GSAT does all necessary updates after a variable has flipped and before another variable is chosen to be flipped, SA randomly chooses a variable after a flip, updates it and, depending on the resulting energy change, possibly flips this variable immediately. So only at first thought, it seems to be correct to take the updates into account in order to compute the amount of

[3] A detailed comparison of several schedules can be found in [1].

[4] Bart Selman, personal communication

work, as argued in [17]. But GSAT uses several algorithmic features reducing the time complexity of an update significantly, and so the comparison becomes difficult. First of all, GSAT uses some data structure which prevents it from doing unnecessary updates of variable scores, a technique we call *score-by-need*. If a variable is flipped, it is not necessary to update the scores of all other variables, but only of those variables, which share a clause with the previously flipped variable. As [17] explains, in a formula with fixed clause length L and a ratio of variables to clauses of C each variable shares clauses with about $L * (L-1) * C$ other variables. So for 3SAT–problems with a ratio of $C = 4.3$ GSAT consequently computes about $3*2*4.3 = 25.8$ new ΔEs for each flip, which is independent of the problem size, instead of recomputing the ΔEs for all variables. As we use a neural network for the SA algorithm, this feature is inherent to our algorithm: Each unit of the network is only connected to those other units which represent variables sharing a clause with the variable represented by this unit. And the score of a variable V is computed with the following formula: Be u_V the corresponding unit, $net\_input(u_V)$ the input of u_V, and $act(u_V)$ its activation (on $= 1$, off $= 0$). Then

$$\Delta E(V) = net\_input(u_V) \cdot (2act(u_V) - 1).$$

So it is easy to avoid updating scores of units which are not connected to a flipped unit and vice versa each units considers only connected units if its score has to be updated. There is no global decision which variable to flip like in GSAT, and consequently it is not necessary to update the scores of all variables after a flip is made.

GSAT also uses another programming "trick" described in [2]: The score of a variable need not be *recomputed* (which has linear time complexity) after each flip but can be updated in constant time[5]. This feature also can be easily used within the SA algorithm. But here lies the problem in comparing the amount of updates performed by GSAT and SA: GSAT does about $n = 25.8$ updates per flip and each of those updates can be done in constant time c_1. So every flip causes $n * c_1$ steps of work in average. During the SA algorithm, between two flips about 5.5 updates were performed. But as between two updates of a variable more than one flip may have occurred, one update may be more costly than in the GSAT case. To compare both algorithms we therefore must compute the average number of steps needed by an update in SA, given that one update in GSAT needs c_1 steps.

Be $f = F/A$ the number of flips per assignment and $u = U/A$ the number of updates per assignment, N the number of variables (units) in the problem. Then each unit is updated about u/N times per assignment and flipped f/N times. The total number of variables affecting a single other unit is about n due to the above considerations. So the total number of flips affecting a unit's score is $f/N * n$ per assignment. Consequently for each update of a unit's score we must consider about $\frac{f/N*n}{u/N} = f/u * n$ flips of connected units. This means that each update needs about $1/5.5 * 25.8 * c_1 = 4.7 * c_1$ steps. To compare GSAT with SA we therefore should multiply the number updates of SA by 4.7 in order to obtain comparable numbers or, very simply, compare the number of flips instead of updates, as in any case we end up with an average amount of $25.8 * c_1$ steps per flip!

Comparing the results of both algorithms, we find that both show about the same success rate (number of assignments found per try). SA needs some more flips, and the ratio of flips between both algorithms seems to increase with the number of variables of the problem. So indeed GSAT with random walk scales up better. But nevertheless one has to keep in mind that the GSAT version is already a very improved one. A comparison with GSAT with *random noise* [15] (which is comparable to the randomness used in SA) shows, that this scales up better that basic GSAT (without any random upwards moves) but is still significantly worse than GSAT with random walk. So it might be possible, that indeed a SA variant using random walk would outperform GSAT. [17] already did experiments with such a SA variant, but unfortunately we do not know how the problems he used were generated, i.e. if his results are comparable with ours. If comparing his results for SA and SA with random walk, we find that SA with random walk shows about the same success rates as simple SA but needs much less flips per assignment. Nevertheless, the ratio between the numbers of flips of both variants decreases with the number of variables of the problem, although staying below 1. On the other hand in our experiments the ratio between the numbers of flips of ordinary SA and GSAT with random walk *increase* with the number of variables. So it is likely that, even when including random walk in both approaches, SA will stay slightly worse than GSAT. In addition, when using a neural network representation for SA, the inclusion of random walk is rather difficult, because the clause structure of the underlying formula gets lost. One would have to provide a special data structure only for this purpose. On the other hand, we gain the advantage of massive parallelism by using neural networks.

4 Related Work

There are two other papers dealing with a comparison of the performance of GSAT and SA on the problem of finding satisfying assignments for random 3CNF-formulas (3SAT). One is the work done by B. Selman, H. Kautz, and B. Cohen [15], the other one is by W. Spears [17]. The two comparisons are totally different as they use different SA algorithms, different variants of GSAT and different ways of comparison. Consequently, also their results differ. An overview of the differences in both comparisons is given in Table 1. Selman, Kautz, and Cohen [15] use GSAT with random walk for the comparison, which scales up significantly better than basic GSAT, whereas Spears [17] uses only basic GSAT for the comparison. In addition, [17] uses the results of [16] which report a significantly higher amount of flips per assignment than new experiments with basic GSAT reported in [15], and he compares basic GSAT with a variant of SA performing also random walk, which scales up even better than SA itself. So the comparison of [17] is unfair in so far as the very poor basic variant of GSAT is compared with a rather "tuned" version of SA including random walk. Interestingly, also [15] report that SA performs significantly better than basic GSAT, but is worse than GSAT with random walk when comparing the numbers of flips. [17] compares also the amount of up-

[5] As [2] point out, this trick speeds up their algorithm by about 100 times. It is not clear why they report a constant speed up, it should be linear in the number of variables of the problem.

dates necessary, as they need also a certain amount of work. This may be correct, if a naive method of *computing* the new ΔEs after a flip is used. But due to the algorithmic improvements used within GSAT, the comparison of updates is still more complicated as we explained in the previous section. In our experiments comparing the simple number of updates for GSAT and SA would also have lead to the conclusion that SA scales up significantly better than GSAT. To summarize, we may say that [15] see GSAT as the "winner" whereas [17] conjectures that SA scales up better. But at least the comparison in [17] seems to be preferential for SA. One could argue that we should compare basic GSAT with SA, as then we have two "original" un-tuned versions of both algorithms. But by comparing GSAT with random walk and SA we see that the improvements of GSAT only lead to an approach to the principles of SA.

	Selman/Kautz/Cohen [15]	Spears [17]
GSAT-Variant	random walk	basic
SA-Variant	general	random walk
SA-Schedule	constant temperature	geometric
max_temp	0.2	0.3
Distribution	Metropolis	Glauber
Comparison	Flips	Updates
"Winner"	GSAT	SA

Table 1. The two comparisons of GSAT and SA

5 Parallelization of GSAT and SA

The question of parallelizability of both algorithms is interesting for at least two reasons. First of all, in the future presumably the most powerful computers will be massively parallel ones. Second, problems occurring in practice will not be always those lying in the "hard" area. As the satisfiability problem is $\mathcal{NP}$-complete, one cannot expect to gain anything by parallelization, if the problem considered really is one of the hard ones. But if it is an easier one, it might be possible, that the performance of parallel algorithms significantly improves over sequential ones (e.g. if we touch a subclass of problems lying in the class $\mathcal{NC}$ of efficiently parallelizable problems). Experiments in this direction still have to be done, but this is why we should take the parallelizability of these algorithms into account.
The parallelization of SA is straight forward, if a neural network representation is used. As each unit in the network (resp. each variable) decides if it is updated or flipped only by means of *local* information, updates can be performed simultaneously in parallel and flips can be performed at least asynchronously. Another possibility would be to use a parallel SA algorithm like the one presented in [2]. Here multiple (locally) possible flips are summarized to a "multi-flip", which is accepted by means of a global decision considering the global score of a synchronous flip of all accepted variables. If at each time only a (always changing) portion of the variables is chosen, the problem of getting trapped in a cycle of recurring states is avoided. This algorithm seems to scale up significantly better than general SA [2]. Unfortunately, up to this moment no theoretical results about its convergence behavior exist, which means that it cannot be guaranteed that the algorithm converges even to a *local* minimum. Nevertheless, experimental results lead to the conclusion that the algorithm indeed shows convergent behavior and also converges to a *global* minimum in the limit.
There are also several possibilities of parallelizing GSAT. Perhaps the most obvious (and naive) idea is to evaluate several tries simultaneously using MAXTRIES processors. But this possibility excludes many variants of GSAT using a kind of memory, e.g. *averaging in* and *clause weights* as described in [14], because those use information about previous unsuccessful tries to choose a new initial assignment or change the energy surface, resp. It is also possible to parallelize scoring using N processors and straight-forward scoring or score-by-need and a number of processors independent of N. Unfortunately, as GSAT needs global information to decide which variables are elements of the `possflips` list, a very high amount of communication would be necessary, slowing down the whole procedure. This could be avoided if an indifferent variant of GSAT is used. As reported by [5], indifference does not lead to significantly worse behavior. They even conjecture that greediness is not important for the performance of GSAT. A third approach (analogous to the parallel SA algorithm introduced in [2]) would be to combine the flips of a subset of *possFlips* to a multi-flip, the execution of which is probabilistically dependent of its score (which of course can be totally different from the scores of the single flips).

6 Conclusion

Seen from a SA point of view, we find, that many features already inherent in SA algorithms are embedded in the refined version of the GSAT–frame we used. Seen from a GSAT standpoint, SA is nothing else than a GSAT variant. We have shown this by formulating both algorithms within the GenSAT–framework in Section 2.
SA algorithms typically use only local information to select the next flip; in this respect, they behave similar to indifferent GenSAT variants [5] which consider all variables with negative score as elements of the *possFlips* list. As [5] argue, greediness is not an important feature for the performance of GenSAT, which may be confirmed by our results showing very little difference in the performance of the greedy GSAT algorithm and the indifferent SA algorithm. Randomness plays an important role in SA, usually probabilistic methods as the Metropolis or the Glauber test [6] are used to decide, whether a given variable is actually flipped or not. The probability commonly depends on two factors: the temperature parameter and the score. Of course, this kind of randomness can be easily achieved in GenSAT; one only has to chose a suitable selection-function for *hillClimb*, as we have shown in Figure 1. Note, that during one execution of GenSAT's inner loop exactly one variable is flipped, whereas annealing-algorithms typically flip a random number of variables per cycle. This discrepancy can be avoided, if GenSAT's *hillClimb* is designed in such a way, that it checks the variables in the *possFlips* list in a fair way (ensuring that given a large number of executions of the inner loop, all variables are checked with the same probability). We then achieve a behavior, which is very similar to that of certain SA algorithms. To modify the temperature-parameter during the run, SA algorithms typically use an annealing schedule. This can also be realized by GenSAT variants: again, *hillClimb* has to be defined in a suitable way,

using GenSAT's loop-variable j as additional parameter.
On the other hand many improvements of GSAT to simply speed up the algorithm are already realized within SA: The avoidance of updating the score of variables which are not affected by the previous flips is inherent to neural networks because of their connectionist structure. The possibility of random upwards moves on the energy surface is one of the main features of SA, introduced later to GSAT to greatly improve its performance.
Above all, there are many features included into GSAT to improve its performance by the time as experience with this algorithm grew up. Most of them are already inherent to each SA algorithm in general, especially if a neural network representation is used. Some other improvements may still be included into SA. So it seems worth to investigate SA further and try to find further improvements as here already the basic algorithm is very powerful compared to the basic GSAT variants (as shown in [17]). We have shown that the principles necessary to improve GSAT lead to an algorithm very similar to the well known SA algorithm, which is not adapted to special problems classes in any way. It therefore will be interesting to further investigate how powerful SA still can be made. In addition, for SA there exists a theoretical convergence result. We do not know about a similar result for GSAT.
As explained in section 5, the question of parallelizability is also very important. As SA is much better suited for parallelization than GSAT, we should investigate further the possibilities of SA. There still remain some very interesting questions to be answered: Can we provide a convergence results for the parallel SA algorithm introduced in [2]? What would be an optimal way to parallelize GSAT? What about the behavior of parallel SA vs. parallel GSAT? We anticipate that SA would scale up better, but experiments still have to be done. How do both approaches behave, if we do *not* consider the hardest problems, but those that are under- or over-constraint? What can we do, if we really do not know if a given formula is satisfiable or not?
Acknowledgements: We thank O. Steinmann and M. Lindner for their support during the experiments, S. Brüning and S. Hölldobler for many helpful suggestions on earlier drafts on this paper, and B. Selman and B. Spears for many helpful explanations on their experiments.

REFERENCES

[1] A. Beringer, G. Aschemann, H. Hoos, M. Metzger, and A. Weiß, 'GSAT and Simulated Annealing – a comparison', Technical Report, TH Darmstadt, AIDA-94-01, 1994.
[2] N. Boissin and J.-L. Lutton, 'A parallel simulated annealing algorithm', *Parallel Computing*, **19**, 859–872, (1993).
[3] J. M. Crawford and L. D. Auton, 'Experimental results on the crossover point in satisfiability problems', in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 21–27. MIT press, (1993).
[4] M. Davis and H. Putnam, 'A computing procedure for quantification theory', *Journal of the ACM*, **7**, 201–215, (1960).
[5] I. P. Gent and T. Walsh, 'Towards an understanding of hill–climbing procedures for SAT', in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 28–33. MIT press, (1993).
[6] R. J. Glauber, 'Time-dependent statistics of the ising model', *Journal of mathematical physics*, **4**, 294–307, (1963).
[7] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company, 1991.
[8] J. J. Hopfield, 'Neural networks and physical systems with emergent collective computational abilities', in *Proceedings of the National Academy of Sciences USA*, pp. 2554 – 2558, (1982).
[9] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, 'Optimization by simulated annealing', Technical Report Research Report RC 9335, IBM Thomas j. Watson Center, Yorktown Heights, N.Y., (1982).
[10] W. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, 'Equation of states calculations by fast computing machines', *Journal of Chemical Physics*, **21**, 1087–1092, (1953).
[11] G. Pinkas, 'Symmetric neural networks and logic satisfiability', *Neural Computation*, **3**, (1991).
[12] G. Pinkas, *Logical Inference in Symmetric Connectionist Networks*, Ph.D. dissertation, Washington University, Sever Institute of Technology, St. Loius, Missouri, 1992.
[13] H. W. Schmidt, 'ICSIM: Initial design of an object-oriented net simulator', Technical Report TR-90-055, International Computer Science Institute, (1990).
[14] B. Selman and H. Kautz, 'Domain-independent extensions to GSAT: Solving large structured satisfiability problems', in *Proceedings of the International Joint Conference on Artificial Intelligence*, ed., R. Bajcsy, volume 1, pp. 290–295. Morgan Kaufmann Publishers Inc., (1993).
[15] B. Selman, H. A. Kautz, and B. Cohen, 'Local search strategies for satisfiabiliy testing', Technical report, AT&T Bell Laboratories, (1993).
[16] B. Selman, H. Levesque, and D. Mitchell, 'A new method for solving hard satisfiability problems', in *Proceedings of the AAAI National Conference on Artificial Intelligence*, pp. 440–446. MIT press, (1992).
[17] W. M. Spears, 'Simulated annealing for hard satisfiability problems', Technical report, Naval Research Laboratory, Washington D.C., (1993).