Adding New Clauses for Faster Local Search

Byungki CHA and Kazuo IWAMA
Dept. of Computer Science and Communication Engineering
Kyushu University, Hakozaki, Fukuoka 812-81, JAPAN
{cha, iwama}@csce.kyushu-u.ac.jp

Abstract

A primary concern when using local search methods for CNF satisfiability is how to get rid of local minimas. Among many other heuristics, Weighting by Morris (1993) and Selman and Kautz (1993) works overwhelmingly better than others (Cha and Iwama 1995). Weighting increases the weight of each clause which is unsatisfied at a local minima. This paper introduces a more sophisticated weighting strategy, i.e., adding new clauses (ANC) that are unsatisfied at the local minima. As those new clauses, we choose resolvents of the clauses unsatisfied at the local minima and randomly selected neighboring clauses. The idea is that ANC is to make the slope of search space more smooth than the simple weighting. Experimental data show that ANC is faster than simple weighting: (i) When the number of variables is 200 or more, ANC is roughly four to ten times as fast as weighting in terms of the number of search steps. (ii) It might be more important that the divergence of computation time for each try is much smaller in ANC than in weighting. (iii) There are several possible reasons for ANC's superiority, one of which is that ANC returns the same local minima much less frequently than weighting.

Introduction

Many people agree that it is one of the most remarkable findings in the 90's that the local search works well for CNF satisfiability testing in spite of the problem's highly combinatorial nature (Gu 1992; Selman *et.al* 1992). That was followed by an even more important finding; the weighting strategy to escape from local minimas (Morris 1993; Selman and Kautz 1993). Cha and Iwama (1995) showed that the weighted local search is overwhelmingly faster than other local search algorithms such as GSAT (Selman *et.al* 1992) and Random Walk (Selman and Kautz 1993). Local search starts from a random truth assignment and moves to a neighboring truth assignment, one after another, in the direction that the number of unsatisfied clauses decreases. Suppose that one gets to a truth assignment, called a *local minima*, where the number of unsatisfied clauses is less than or equal to the number of unsatisfied clauses at any neighboring truth assignment. Then the weighted local search, or WEIGHT for short, increases the weight of each unsatisfied clause at the local minima, hoping that the weighted sum of unsatisfied clauses will be larger than some neighboring assignment and we will be able to escape from the local minima. One good point of this approach is that we do not have to be nervous about details of how to increase the weight. Due to Cha and Iwama (1995), WEIGHT is surprisingly stable against different ways of increasing the weight, integers or reals, fixed values or random values, etc. Ironically this means that there is little hope of improving the weighted local search by simply changing the weighting method.

In this paper, we do try to improve the weighted local search from a little different angle. One can see that increasing the weight of some clause by, say, one, is equivalent to adding one same clause to the formula. Such an added clause may not be one of the existing clauses but may be a new one. The problem is of course how to select this new clause. Our solution here is a resolvent obtained from the unsatisfied clause at the local minima and its neighboring clause. The new algorithm is called ANC (Adding New Clauses). When it gets to a local minima, ANC generates as many such resolvents as it obtains the same effect as WEIGHT increases the weight of the clauses.

Experimental data show that ANC is considerably faster than WEIGHT. Experiments were conducted using random 3CNF formulas (Clause/Variable ratio=4.3) and the 3CNF formulas of C/V ratio 2.0 that have only one satisfying truth assignment. The latter formulas can be generated by the AIM Generator (Asahiro *et.al* 1993) and have shown to be hard and appropriate for local search algorithms (Cha and Iwama 1995). When the number of variables is 200 or more, ANC is four to ten times faster than WEIGHT. Experiments were also conducted for more natural formulas on VLSI design and Block world planning, for which ANC is also better generally. It might be more important that the divergence of cell moves in each try for different formulas or even for the same formula is much smaller in ANC than in WEIGHT. The number of cell moves in each try is within three times the average in ANC but is 10 times or even more occasionally in WEIGHT.

ANC is not a result of trial and error but is due to an intensive analysis of the weakness of WEIGHT, which will be described in Sec. 4. Among others, the most serious problem of WEIGHT is that it tends to visit the same truth assignment many times just as wandering in circles. There are reasonable explanations for why adding the resolvents is effective to prevent this circle-wandering. In fact the number of revisited truth assignments is 10 to 100 times less in ANC than in WEIGHT.

By a rough estimation, the number of cell moves of ANC triples when the number of variables doubles. This means the growth of the number of steps is approximately $O(n^{1.6})$. We need more time in each cell move to add resolvents in ANC than increasing the weight in WEIGHT. However, this portion of computation has a lot of room for improvement by better coding techniques and parallelization.

Weighted Local Search

A *literal* is a (logic) *variable* x or its negation $\overline{x}$. A *clause* is a sum of one or more literals. A (CNF) *formula* is a product of clauses. A specific assignment of *true* (or 1) and *false* (or 0) to all the variables is called a *cell*. (This might be unusual; note that the terminology in this paper is associated with the Karnaugh map being popular in switching theory.) For cells C_1 and C_2, the *Hamming distance* between C_1 and C_2 is the number of variables for which the assignment is different (true for C_1 and false for C_2 or vice versa). C_1 is a *neighbor* of C_2 if their Hamming distance is one. It is said that a clause A *covers* a cell C if the truth assignment denoted by C makes A *false*.

The number of overlaps of a cell C, denoted by $OL(C)$, is the number of the clauses that cover C. A cell C is called a *local minima* if $OL(C') \geq OL(C)$ for all neighbors C' of C. A cell is called a *solution* if $OL(C) = 0$.

Basically, local search for CNF satisfiability tries to reach a solution by gradually moving to cells of a smaller number of overlaps hoping that it can eventually get to a cell of zero overlaps. Its fundamental structure is as follows:

```
**Algorithm** Local_Search
  $C$ := a randomly selected initial cell
  **until** $C$ becomes a solution **do**
  **if** $C$ is not a local minima **then do** $\alpha$ **else do** $\beta$
```

A specific local search algorithm is obtained by completing α and β. For example, in the simplest algorithm called RESTART, α = Move-downward that is to move the one of the neighboring cells whose overlaps (less than the current one) are the least and β = Restart that stops the current search and goes to a new initial cell. In the well-known GSAT (Selman *et.al* 1992), $\alpha = \beta$ = Move-to-minimum (moving to the neighboring cell whose overlaps are minimum regardless of being less than the current overlaps) + Restart at every fixed number of steps.

In weighted local search algorithms, a *weight*, $w(A) \geq 0$, is associated with each clause A. Its value is initially one for all clauses. Now the definition of the number of overlaps is changed into the sum of the weights of the clauses that cover the cell. Namely

$$OL(C) = \sum_{A \text{ covers } C} w(A)$$

In the standard weighted local search, *WEIGHT*, α = Move-downward and β = Weighting, i.e., to add one to $w(A)$ for all the clauses A which cover the current cell. As shown in Cha and Iwama (1995), WEIGHT is overwhelmingly faster than others. The difference of performance can reach as large as 100 times even when the number of variables is moderate, say, 100.

Recall that WEIGHT adds a unit weight (one) to each clause covering the current cell. Different weighting strategies include: (i) adding two to each clause, (ii) adding a real value to each clause that is minimum enough to escape from the current local minima, (iii) adding a random value between 0.5 and 1.5 to each clause and (iv) adding one to a single clause selected at random. It is a little surprising that the performance of the weighted local search is very similar for their different weighting strategies (Cha and Iwama 1995).

Another important feature is that WEIGHT works much better for high-C/V-ratio formulas than for low-C/V-ratio ones. This is claimed in Cha and Iwama (1995) using the instances such that they all have the same number (= one) of satisfying truth assignments to make conditions even. There is at least one obvious reason for the better performance: When the C/V-ratio is high, the average number of clauses covering each cell is high and there is a nice "slope" down to the solution (= 0 overlaps) which begins from a fairly distant place from the solution.

Adding New Clauses

One can see that increasing the weight of a clause A is equivalent to adding another same A to the formula f (i.e., two A's exist in f). This leads to the idea that we can expect a similar effect as weighting by adding new clauses. The problem is how to find the new clauses. There are two necessary conditions: (i) they must not cover the satisfying cells. (ii) They should increase the number of overlaps at the local minima. Our solution is to select resolvents of two clauses.

For a clause A_1, another clause A_2 is called a *neighboring* clause, or A_1 and A_2 *are neighboring*, if there exists exactly one variable x which appears affirmatively in A_1 and negatively in A_2 (or vice versa). For example, $(x_1 + x_3 + \overline{x}_4)$ and $(x_1 + \overline{x}_3 + x_5)$ are neighboring. For two neighboring clauses $A_1 = (x + X)$ and $A_2 = (\overline{x} + Y)$, the clause $(X + Y)$ is called a *resolvent* of A_1 and A_2. For example, $(x_1 + \overline{x}_4 + x_5)$ is the resolvent for the previous two clauses.

```
Algorithm ANC
  $S_Q$ := the set of clauses in the given formula $f$
  $S_A$ := $\phi$
  $C$ := a randomly selected initial cell
  until $C$ becomes a solution do
  if $C$ is not a local minima
     then do Move-downward
     else for each clause $A$ in $S_Q$ which covers $C$,
        do find a neighboring clause $B$ in $S_Q \cup S_A$
              such that the resolvent $X$ of $A$ and
              $B$ covers $C$; add $X$ into $S_A$ (Usually
              there are many such $B's$, one of
              which is selected at random. If no
              such $B$ exists, than no resolvent is
              added for the $A$, which, however
              happens very rarely.)
```

Recall that WEIGHT increases a unit weight for each clause covering the local minima. Instead, ANC adds one resolvent for each clause covering the local minima if we assume that such a resolvent can always be found. Theoretically, there is a chance that we cannot find an enough number of resolvents to leave the local minima; the algorithm falls into a cycle. We have never experienced that this actually happened so far. If one wishes to avoid this more certainly, here is a simple modification: As the clause x added in S_A, the clause A itself is allowed if there is no resolvents. Another possible modification is that not only a clause in S_Q but a clause in S_A is also considered to take a resolvent with B. We actually experimented this method, but there were no clear differences.

Experiments were conducted using two different types of 3CNF formulas: One is completely random formulas of C/V-ratio 4.3 (but only satisfiable ones) and is denoted by rN, where N is the number of variables such as $r100$. The other is the formulas generated by AIM Generators (Asahiro *et.al* 1993) which are basically random-instance generators but can control several attributes of generated instances including C/V-ratio and the number of solutions. In this paper, we used the formulas whose C/V-ratio is 2.0 and that have only one solution. Probably they are hardest test-instances for local search algorithms that are available in the form of random instances (Asahiro *et.al* 1993). This type of instances are denoted by oN such as $o100$.

Table 1 shows the performance of ANC and WEIGHT. In this paper the performance is always shown by the number of cell moves. Without others stated, each figure is the average of 100 instances. One can see that ANC is significantly better than WEIGHT. $o400$ is quite hard, for which only two instances were able to be solved by ANC and WEIGHT within a reasonable time. The last three instances are not random formulas but more natural ones on fault diagnosis of VLSI design, etc. As for these three formulas, we ran the algorithm five times for each. The figures on the table are their averages.

Instances	ANC	WEIGHT
$o100$	1510	5508
$o200$	6108	86185
$o400$	14316	440198
$r100$	206	299
$r200$	823	7555
$r400$	1930	8058
$r800$	3845	19423
2bitadd_12.cnf	330	1222
2bitcomp_5.cnf	65	409
3blocks.cnf	6097	1488

Table 1: Performance of ANC and WEIGHT

Figs. 1 to 4 show the divergence of the performance for each try. For example, the upper-left graph in Fig. 1 shows that among 100 instances (100 tries), two instances need less than 500 steps, nine 500-1000 steps, and so on. Figs. 1 and 2 show the divergence for different instances and Figs. 3 and 4 the divergence for different (100) tries against the same instance. In both cases, divergence is larger in WEIGHT than in ANC. This tendency is especially clear in one-solution instances. WEIGHT occasionally encounters "extremely slow tries" which are up to 20 times slower than the average even for those instances of relatively small size. It turned out that this becomes more serious as the size of instances grows. The local search is basically an incomplete algorithm in the sense that it cannot be used to show that the formula is unsatisfiable. However, as claimed in Selman *et al.* (1992), it may be a hint of unsatisfiability that the search procedure does not stop for sufficiently long time. For this purpose, a small divergence of the running time is truly convenient.

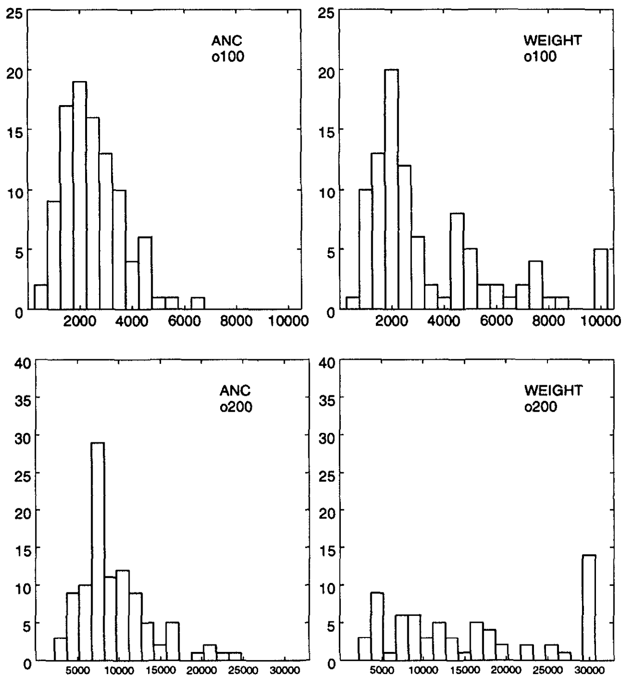

Figure 1: Divergence of the Performance

Why and How ANC Works

This section is for intensive discussion of what kind of weak points are in WEIGHT and why ANC can compensate them. We shall first discuss the problem of revisiting the same cells. The second topic is the size of added clauses. Note that a clause including three literals covers one eighth of the whole search space (= the whole cells). Considering that the purpose of adding clauses is to "fill a hole" that exists at a local minima, this size of the added clause seems to be too large.

• Revisiting Same Cells

As briefly mentioned in Cha and Iwama (1995), WEIGHT often visits the cells that have already been visited. It is also known that there is a strong correlation between the amount of those revisited cells and the number of search steps; when there are many revisited cells, WEIGHT takes long time. In what follows, we shall take a detailed look at this behaviour, what

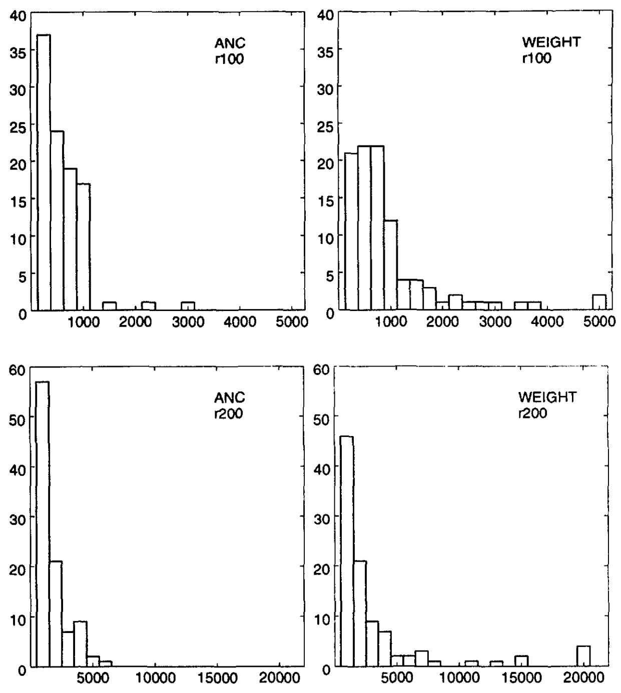

Figure 2: Divergence of the Performance

the revisit looks like at the level of each cell-move, why that happens in WEIGHT and what can possibly prevent that.

Fig. 5 illustrates how the revisit occurs in WEIGHT. This figure shows cell-moves of WEIGHT for some $o100$ instance from the 2601st step to the 2669th step. Each circle means a cell and the number in it shows in what step WEIGHT comes there for the first time. Arrows show how WEIGHT moves between cells. Thus, during these 69 steps, only 19 different cells are visited. A much more important fact is that the Hamming distance between each of those cells and the cell numbered 2602 is at most four. Namely, if we consider the 2602 cell as a center, the 67 moves are within a circle of only radius four. This is what we call "wandering in circles".

To consider the reason for this behaviour, we shall show another data: Suppose that a cell C is a local minima. The number of overlaps there is usually quite few, typically three through five according to not only our experiments but others (e.g., Selman and Kautz 1993). Then how many different clauses are there which cover some neighboring cell of C? It is quite stable again according to experiments; about a half of the whole clauses (say, about 100 of the instance is $o100$). Note that each of the neighboring cells of C is usually covered by a different clause among these

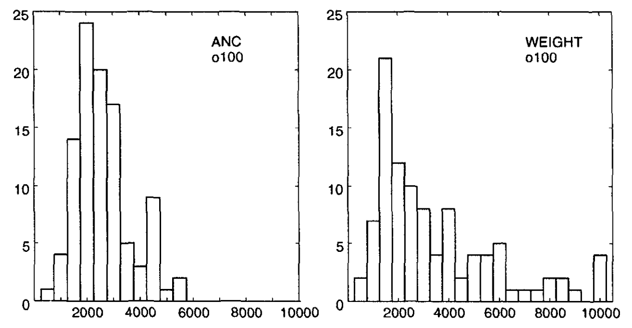

Figure 3: Divergence of the Performance

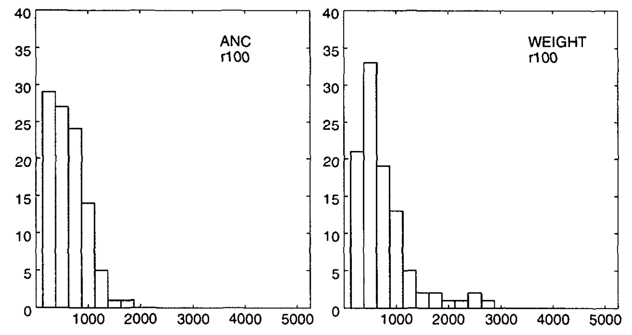

Figure 4: Divergence of the Performance

neighboring clauses.

Thus the reader can imagine what is illustrated in Fig. 6, where A_0 is the clause covering the local minima C_0 and clauses A_1, A_2, ... cover different neighboring cells C_1, C_2, ..., respectively. Now what happens if we raise the height of A_0 by weighting. Then, WEIGHT will probably move to C_1. Then if C_1 is again a local minima, there is a great chance to move back to C_0 by raising the height of C_1. Even if it is lucky enough to move other cells, C_2, C_3, and so on, it is just like traversing on a contour at nearly the bottom of the hole (= the local minima). It is likely that those cells are relatively close to C_0 and again there is a good chance to reverse the traverse or just to go down to C_0 again.

This observation leads us to the conclusion that it is probably better to raise the portion which spreads over both A_0 and A_1, than to raise A_0 only. One can see that the resolvent of A_0 and A_1 exactly satisfies this condition. Table 2 shows how the number of revisited cells decreases by ANC. This result is remarkable and is probably the reason why ANC does not encounter the extremely slow instances.

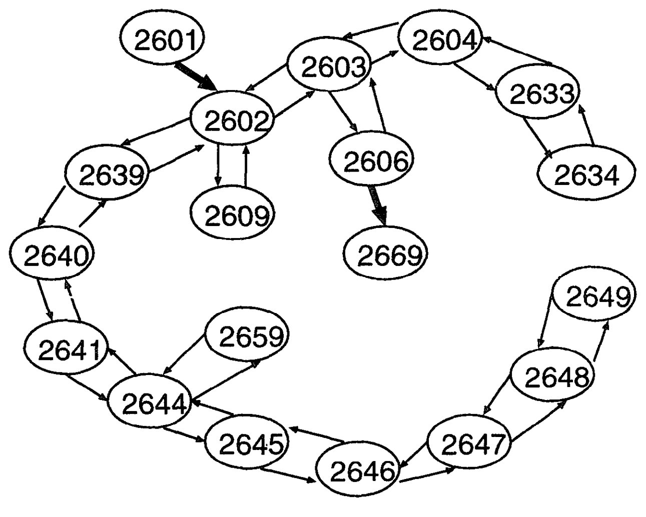

Figure 5: Revisiting Same Cells in WEIGHT

• Decreasing the Size of Added Clauses

Although we mentioned that a clause of three literals is too large "to fill the hole", it is a difficult question to ask proper size. One hint might be the size of the hole or the local minima. Fig. 7 shows the average number of overlaps vs. the Hamming distance from the local minima. In the case of $o100$ instances, the average number of overlaps is 25, so the radius of the hole is roughly 35 in terms of the Hamming distance. If we assume that this is a circle, the number of cells included in it is $2^{100} \cdot \sum_{i=1}^{35} \binom{100}{i} \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{100-i} \approx 2^{91}$. This is much smaller than 2^{97} that is the number of cells covered by a clause of three literals.

To see the effect of adding smaller-sized clauses, we conducted experiments of the algorithm that is the same as ANC but the added clause is not the resolvent but the clause which is obtained by adding one random literal into the clause covering the local minima (i.e., the size becomes one half). The result is shown in Table 2. Where the new algorithm is denoted by HALF. As one can see, HALF is better than WEIGHT.

Instances	HALF	ANC	WEIGHT
$r100$	786	297	1222
$r200$	1565	1217	3303

Table 2: Performance of Algorithm HALF

In the case of ANC as well, the size of added clauses is likely to be smaller than clauses of three literals. The reason is that the resolvent is taken between a clause of three literals (in S_Q) and a clause of three

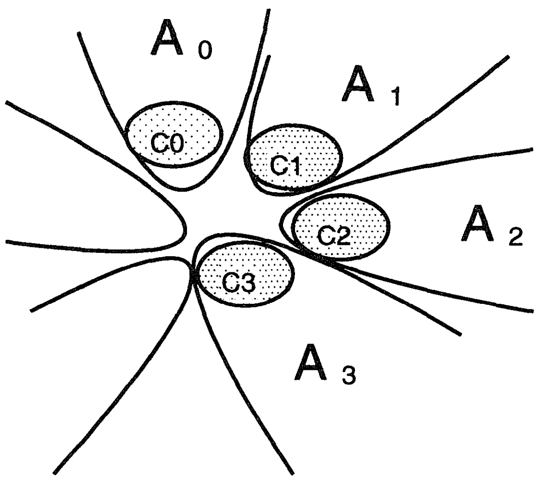

Figure 6: Illustration of the Local Minima and Its Neighbors

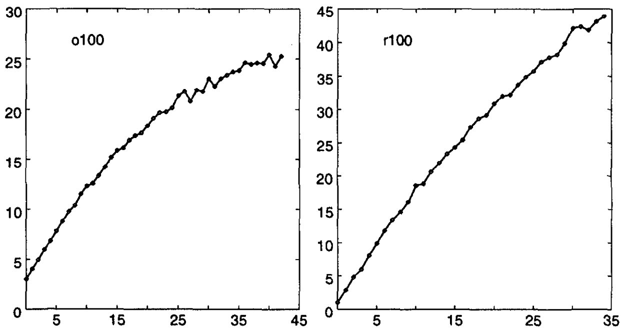

Figure 7: Average Overlaps around the Local Minima

or more literals (in $S_Q \cup S_A$). To make this clearer, we calculated the total number N_c of cells which are covered by the whole added cells. Table 3 shows this number for each try of ANC and WEIGHT where each entry is a pair of the value of $N_c/2^{97}$ and the number of total search steps. As one can see, we selected the tries in which the number of search steps is roughly the same for ANC and WEIGHT. Nevertheless, the total number of the cells covered by added clauses is much less in ANC than in WEIGHT; N_c of ANC is about the half of N_c of WEIGHT algorithm.

Concluding Remarks

The analysis given in Sec. 4 suggests several other possibilities for improvements of the weighted local search:

(1) Further smaller size of added clauses may be even better. To make the added clause further smaller, one can take a resolvent once more with another clause.

(2) ANC currently takes the resolvent between the clause covering the local minima and a randomly selected neighboring clause A. We can impose some conditions on this clause A, e.g., the one that covers some neighboring cell of the local minima.

Instances	ANC	WEIGHT
o100-1	(1233,2333)	(2230,1828)
o100-2	(1504,3010)	(4238,2909)
o100-3	(1879,3417)	(15450,9656)
r100-1	(19,57)	(29,54)
r100-2	(128,229)	(255,234)
r100-3	(284,472)	(534,430)

Table 3: The Number of Cells Covered by Added Clauses

In this paper, the performance is measured only by the number of cell moves. However, we cannot ignore the computation time needed in a single cell-move which is especially important for algorithms having complicated cell-move strategies, like ANC. In practice it is very important to reduce this portion of computation time using cleaver data structure, randomization techniques (e.g., computing the overlaps of not every neighboring cell but some randomly selected ones), and so on. It might also be possible to make use of supercomputers.

References

Asahiro, Y., Iwama, K. and Miyano, E. (1993). Random generation of test instances with controlled attributes, *2nd DIMACS Challenge Workshop.*

Cha, B. and Iwama, K. (1995). Performance test of local search algorithms using new types of random CNF formulas, *Proc. IJCAI-95*, Vol.1, pp.304-310.

Dubois, O., Andre, P., Boufkhad, Y. and Carlier, J. (1993). SAT versus UNSAT, *2nd DIMACS Challenge Workshop.*

Gu, J. (1992). Efficient local search for very large-scale satisfiability problems, *Sigart Bulletin*, Vol.3, No.1, pp.8-12.

Morris, P. (1993). The breakout method for escaping from local minima, *Proc. AAAI-93*, pp.40-45.

Selman, B. and Kautz, H.A. (1993). An empirical study of greedy local search for satisfiability testing, *Proc. AAAI-93*, pp.46-51.

Selman, B. and Kautz, H.A. (1993). Local search strategies for satisfiability testing, *2nd DIMACS Challenge Workshop.*

Selman, B., Levesque, H.J. and Mitchell, D.G. (1992). A new method for solving hard satisfiability problems, *Proc. AAAI-92*, pp.440-446.