A SAT Approach for Solving the Staff Transfer Problem

S. Acharyya, A. Bagchi,

***Abstract*—An important issue in Human Resource Management is the assignment of transfer postings to employees in a large organization that has offices and worksites at multiple locations. It is customary in such organizations to transfer a subset of employees at periodic intervals. This practice lends significance to the Staff Transfer Problem (STP), which can be viewed as a type of Constraint Satisfaction Problem (CSP). Deterministic methods for solving the STP are demonstrably inferior to local search methods. In our earlier investigations we found that Simulated Annealing (SA) performed the best among local search techniques. But our recent computer experiments indicate that an improved GSAT formulation implemented using tabu lists can outperform SA in many situations.**

***Index Terms*— Constraint Satisfaction, Simulated Annealing, Satisfiability, Tabu Search, Staff Transfer**

I. Introduction

A transfer is a lateral movement of an employee in an organization not involving a change in rank. In many large organizations that maintain offices and worksites at multiple locations, it is customary to transfer a subset of employees at periodic intervals from one office or worksite to another. Examples of such organizations are the armed forces, government departments, commercial banks, and construction firms. Transfers play a major role in human resource flow in organizations, and the satisfactory assignment of transfer postings to employees is an important issue in Human Resource Management [4,5].

The *Staff Transfer Problem* (STP) [1] can be formulated as a Constraint Satisfaction Problem (CSP). The variables take values that correspond to transfer postings of employees. A typical constraint enforces the requirement that the total number of employees assigned to a transfer posting should not exceed the specified number of vacancies. The objective is to find transfer postings for a given subset of employees that satisfy all the constraints.

The constraints here are not binary but involve a number of variables. The constraints are of two types, *strict* and *desirable*. A strict constraint must be satisfied; for example, an employee can only be transferred to a location where there is a vacancy for a job the employee can perform. A solution that satisfies all strict constraints is a *feasible solution*. A desirable constraint should be satisfied if possible. For example, it is desirable that senior employees get priority over their juniors in the assignment of transfer postings. The quality of a feasible solution is determined by the degree to which desirable constraints are satisfied. The objective is to find feasible solutions of high quality.

The transfer policy that is adopted in an organization must be both fair and systematic to be acceptable to employees. When the transfers involve a large number of employees, it is typically quite difficult to find suitable alternative positions for everyone. Some preliminary results are reported in [2]. More detailed results comparing various CSP techniques and showing the superiority of Simulated Annealing (SA) over other methods are reported in [3]. In earlier investigations, when the Staff Transfer Problem (STP) was partially converted to a Satisfiability Problem (SAT), its performance was inferior to that of a Simulated Annealing (SA) formulation. In this paper we explain how the STP can be fully expressed in SAT. The resulting implementation of GSAT with a tabu list outperforms SA. This approach is interesting for two reasons:

i) We do not need to assign costs to the constraints. In SA, the search is directed by the cost function.
ii) Constraints are directly converted to clauses. We get feasible solutions of high quality when all the clauses are satisfied.

Sec 2 below describes the Staff Transfer Problem (STP), and Sec 3 presents some solution methods. Experimental results are provided in Sec 4. Sec 5 summarizes the paper and discusses some unsolved issues.

II. Problem Description

There is a set of E employees who are to be transferred. The organization has other employees who are not being transferred. At any office or worksite, the number of vacancies is the difference between the number of sanctioned positions and the number of employees who are not being transferred. An employee can perform only certain types of jobs. An employee has a seniority level, which is determined by factors such as the level of the current position in the organizational hierarchy, the date of joining the current position, and the date of birth. There is a set LOC of locations. A location corresponds to an office or a worksite. There is a set J of job categories. Typical job categories are electrician, typist and manager. For each pair (loc,j), loc in LOC and j in J, the number of vacancies *vacancy(loc,j)* is known. This number includes the chain vacancies that arise as a result of the E transfers. It can be zero, indicating that job type j does not exist at location loc or that there is no vacancy for job type j at location loc.

An employee who is due for transfer submits options for $T \geq 2$ destinations, each represented by a (loc,j) pair. The job types in the transfer options correspond to jobs that the

S. Acharyya, West Bengal University of Technology, Kolkata, India, email1: srikalpa8@yahoo.co.in, email2: sriyankar.acharyya@wbut.ac.in
A. Bagchi, Indian Institute of Science Education and Research, IIT Kharagpur Extension Centre, Kolkata, India, email: bagchi@iimcal.ac.in

employee can perform. The parameter T has a typical value of 3. The options are arranged by preference, the first option having the highest preference for the employee, the second the next highest, and so on.

For convenience, we assume that the employees to be transferred are numbered 1 through E in decreasing order of (rank, seniority). Thus employee 1 has the highest rank and is senior-most, and employee E has the lowest rank and is junior-most. Let *vac* be the total number of vacancies. Then vac = E*(1+extra), where *extra* gives the total number of positions lying vacant prior to the transfers, expressed as a fraction of E. It is possible that extra = 0, in which case the chain vacancies are the only vacancies. To avoid problems in implementation arising out of the chain vacancies, we assume that all E transfers take effect at the same instant of time. Let us call each (loc,j) pair a *bucket*. The buckets are numbered 1 through B. The number of vacancies in bucket k = (loc,j) is M_k = vacancy(loc,j).

As already mentioned, constraints are either strict or desirable. There are three strict constraints:

SC1: Each of the E employees must be assigned a new posting. A new posting is a (loc,j) pair that differs from the original (loc,j) pair in at least one coordinate.

SC2: An employee can only be transferred to one of the T options specified by the employee.

SC3: For each (loc,j) pair, the total number of persons transferred to (loc,j) must not exceed vacancy(loc,j).

The problem can be formulated mathematically as follows. An ExTxB matrix $A = [a_{ijk}]$ is given, where $a_{ijk} = 1$ if transfer option j of employee i is bucket k, and 0 otherwise. We have to determine the ExT solution matrix $X = [x_{ij}]$, where $x_{ij} = 1$ if employee i is assigned transfer option j, and 0 otherwise. An employee is assigned exactly one new transfer option. The solution is *feasible* if the number of employees transferred to any bucket k does not exceed the number of vacancies M_k. This condition can be expressed as follows:

$$\Sigma\Sigma\, x_{ij}a_{ijk} \leq M_k \text{ for } \quad 1 \leq k \leq B$$

where the summations are over $1 \leq i \leq E$, $1 \leq j \leq T$. A feasible solution satisfies all the three strict constraints mentioned above.

There are two desirable constraints:

DC1: Among the T options, an employee should be assigned an option of as high a preference as possible.

DC2: When two employees contend for the same transfer posting, the employee who is senior should get priority over the employee who is junior.

DC1 can be expressed by the condition

$$\Sigma\, x_{ij} \geq \Sigma\, x_{ij+1} \quad \text{for } 1 \leq j < T,$$

where the summation is over $1 \leq i \leq E$. This says that more employees should be assigned an option of preference j than an option of preference j+1.

DC2 is satisfied if *seniority violations* can be avoided. Such a violation occurs when there is a senior employee who wants a particular (loc,j) posting with a certain preference, a posting of higher preference being currently unavailable to him (or her). But a junior employee who wants the same posting with an equal or lower preference is assigned the posting instead. This can be stated more formally as follows: A seniority violation occurs if there exist employees k_1 and k_2, where $k_1 < k_2$, and integers r, s and t, $1 \leq s \leq r < t \leq T$, such that:

i) the junior employee k_2 is assigned a transfer option (loc,j) that has preference r for k_2;
ii) the senior employee k_1 is assigned an option of preference $t > r$;
iii) k_1 has specified the same (loc,j) pair assigned to k_2 as an option with preference $s \leq r$.

The condition for no seniority violation, of an employee i assigned option j, by a junior employee q assigned option r is

$$\Sigma\Sigma\Sigma\Sigma\Sigma\Sigma\, x_{ij}a_{ipk}x_{qr}a_{qrk} = 0$$

where the summations are over $1 \leq i < q \leq E$, $1 \leq p,r < j \leq T$, and $1 \leq k \leq B$. In explanation, we note that: i) the junior-most employee cannot suffer a seniority violation; ii) it must be the case that $j > 1$, for if employee i gets the first transfer option then no seniority violation can occur; and, iii) q must get an option $r < j$ to violate the seniority of i. Stated in this form, the Staff Transfer Problem is just a CSP and not an optimization problem, since there is no objective function to optimize.

We can view the Staff Transfer Problem, with seniority violations ignored, as a problem in which a complete matching of maximum weight must be found in a weighted bipartite graph [7]. Let an instance of the Staff Transfer Problem be given in which all bucket sizes are unity. We construct a bipartite graph $G = (V_1 \cup V_2, E')$, where V_1 is the set of vertices on the north side, V_2 the set of vertices on the south side, and E' is the set of edges. V_1 and V_2 are assumed to be disjoint. In our case, V_1 represents the set of employees and V_2 the set of buckets. An edge from a vertex v_1 of V_1 to a vertex v_2 of V_2 indicates that v_2 is a transfer option of employee v_1. We assign to each edge a positive integer weight that depends on the employee, the bucket, and the preference of the transfer option. This weight reflects the value of this particular option to the organization and to the employee. A transfer option that is a first preference has a higher weight than one that is a second preference, and so on. Since the buckets have unit size, our objective here is to find a matching that is complete for V_1 and maximizes the sum of weights of the selected edges. This problem can be solved in time that is polynomial in the size of the input. Now, suppose there is a bucket that has a size $k > 1$. In V_2 we split this big bucket into k smaller buckets each of unit size. If the big bucket happens to be a transfer option with weight w for employee v_1, we join v_1 to each of the k smaller buckets with edges of weight w. As a result we are again left with a complete matching problem. Since bucket sizes are constant integers supplied as input, the solution can still be obtained in time that is polynomial in the size of the input.

When seniority violations must be resolved, it becomes much harder to assign transfer options to employees in a satisfactory manner. It is not yet known whether the Staff Transfer Problem becomes NP-complete in this case.

The Staff Transfer Problem models an idealized situation. In practice, an organization might want to impose additional conditions on the transfer options of employees or change some of the assumptions. For example, the number T of transfer options need not be the same for all employees. The organization could decide how many transfer options an employee would be entitled to submit. Assuming an employee is allowed to submit options freely, it is advantageous to submit as few options as possible, since this means the employee is more likely to get a desired posting. If the organization wants to impose some degree of control over the transfers, it could itself supply one of the transfer options,

perhaps the one of highest preference, and the employee could be requested to submit the remaining options. There are many other possibilities.

III. Solution Methods

A satisfactory analytical solution procedure for the problem, even in its idealized form, has not yet been found. We are thus forced to use heuristic methods. In our earlier work [2,3] it was shown that Simulated Annealing outperformed other heuristic and deterministic methods. Here, apart from using Simulated Annealing, we have tried to solve randomly generated problem instances using GSAT(L). The implementations given below consider only the basic version of the problem consisting of: i) the strict constraints, and ii) the desirable constraint DC2.

A. Satisfiability

The Staff Transfer Problem (STP) can be viewed as a SAT. One way to express the STP in terms of logical satisfiability is as follows. Let the T transfer options of employee k be (loc_1,j_1), (loc_2,j_2), ..., (loc_T,j_T). Then employee k must be assigned a transfer posting at one of these T destinations. This can be mathematically expressed by the logical proposition

$$X(k,loc_1,j_1) \cup X(k,loc_2,j_2) \cup ... \cup X(k,loc_T,j_T)$$

where $X(k,loc_i,j_i)$, $1 \leq i \leq T$, are logical variables that correspond to positive literals. The literal $X(k,loc_i,j_i)$ is *true* if k is assigned bucket (loc_i,j_i), and is *false* otherwise. There are as many such propositions as there are employees. We refer to this set of clauses as the first set.

Another set of clauses arise out of the size restrictions on the (loc,j) buckets. Suppose m' employees $k_1, k_2, ..., k_{m'}$, have all given bucket (loc,j) as the transfer option. If m' > M = vacancy(loc,j), bucket(loc,j) could become overfull. To prevent the bucket from becoming overfull, for *every* subset $\{ k'_1, ..., k'_M, k'_{M+1} \}$ of size M+1 of the set $\{ k_1, k_2, ..., k_{m'} \}$, we generate a clause of the form

$$\sim X(k'_1,loc,j) \cup \sim X(k'_2,loc,j) \cup ... \cup \sim X(k'_{M+1},loc,j)$$

We refer to this set of clauses as the second set. The number of such clauses for this particular bucket is C(m',M+1), the number of combinations of m' objects taken (M+1) at a time.

Seniority violations can be incorporated in our scheme as follows. Suppose a seniority violation has occurred between employees k_1 and k_2, where $k_1 < k_2$. Then, as stated before, there exist integers r, s and t, $1 \leq s \leq r < t \leq T$, such that: i) the junior employee k_2 is assigned a transfer option (loc,j) which has preference r for k_2; ii) the senior employee k_1 has specified the same (loc,j) pair as an option with preference $s \leq r$; iii) k_1 is assigned an option of preference t > r. We can express this constraint as a clause

$$\sim X(k_1,loc_1,j_1) \cup \sim X(k_2,loc_2,j_2)$$

where option t corresponds to (loc_1,j_1) and option r corresponds to (loc_2,j_2). To generate all such clauses, the transfer options of all the E employees must be examined and potential seniority violations identified. We refer to this set of clauses as the third set. Many such clauses would be generated in a typical problem instance.

The satisfaction of the first set of clauses ensures that each employee is assigned a new posting. This means that strict constraints SC1 and SC2 are satisfied. The satisfaction of the second set of clauses ensures that there is no overfull bucket, so strict constraint SC3 is satisfied. The satisfaction of the third set of clauses ensures that there are no seniority violations, so desirable constraint DC2 is satisfied. No effort was made to satisfy desirable constraint DC1, because this constraint is hard to state in terms of clauses.

The greedy local search procedure GSAT initially assigns random truth values to variables in an effort to satisfy all the clauses [11,12]. GSAT looks for the variable with the property that its truth-value when flipped causes the *largest net decrease* in the number of unsatisfied clauses. It flips the truth-value of this variable, and again looks for such a variable. Ties are resolved arbitrarily. This is repeated until a satisfying assignment is found. If no satisfying assignment is found within a specified number of flips (*maxflips*), the procedure is restarted with a new random initial truth assignment. This cycle is repeated a specified number of times (*maxtries*).

The performance of GSAT can be significantly improved by incorporating a *tabu search* strategy to ensure that the same variable is not flipped again and again [10]. The tabu list is initially empty and is implemented as a FIFO queue. A variable that has just been flipped is inserted into the list. The variable to be flipped next is selected randomly [6] from among those variables *not* in the tabu list that cause the largest decrease in the number of unsatisfied clauses. As new variables get flipped and enter the tabu list, older entries fall out of the list at the other end. Thus some variables are prevented from being flipped for a limited period of time, determined by the length of the tabu list. In some applications this length plays a critical role in the performance of GSAT. We represent GSAT with a tabu list of length L as GSAT(L). In Procedure GSAT(L), the values of maxflips and maxtries are selected by trial and error so that good solutions are obtained in reasonable time. GSAT outputs feasible solutions that are *completely free* of seniority violations, so these solutions are superior to those obtained by SA.

Procedure GSAT

```
{
  for (try = 1; try ≤ maxtries; try++) {
    X = TA;                    /* TA gives the initial random truth values of the variables */
    for (flip = 1; flip ≤ maxflips; flip++) {
      if (X satisfies all clauses) return X;
      else {
          determine the set of variables in X flipping which cause the maximum decrease in the number of unsatisfied clauses;
          select a variable from this set resolving ties arbitrarily;
          flip the selected variable;
      }
    }
  }
  announce failure;            /* no satisfying truth assignment found */
}
```

B. Simulated Annealing

The formulation of Procedure SA can be found in [8,9]. An initial trial solution S is obtained by randomly assigning each employee to one of the T buckets corresponding to the given transfer options of the employee. As a result, some of the (loc,j) buckets become *overfull* and have more than vacancy(loc,j) employees. We must now move employees

from the overfull buckets to those that are still not filled up. We randomly select an overfull bucket, randomly select an employee from that bucket, and then place the employee randomly in one of the buckets corresponding to the remaining T-1 transfer options. The performance of the algorithm improves markedly if a tabu list of employees is maintained. The choice of the cost function plays a critical role in the success of the method in finding a solution of good quality. Even when there are no overfull buckets, we can continue generating new trial solutions to find one that satisfies more of the desirable constraints. In this case, instead of selecting an overfull bucket, we randomly select any bucket and proceed as above. The algorithm outputs the feasible solution of lowest cost that it generates.

Procedure SA makes use of a number of parameters. The values of these parameters must be finely tuned, otherwise, inferior results are obtained. The most important issue is the initialization of the temperature and the determination of the rate at which it should decrease. A very high temperature such as 10^6 (one million) is initially chosen. Whenever changes/trials $\geq$ *tcent*, the temperature is halved; if changes/trials < *tcent*, the temperature is reduced slowly; the reduction factor *tempfactor* typically has a value of 0.95. The length of the tabu list can be chosen to be around 3% to 5% of E. Small changes in the length do not have much effect on the runtime, but the performance deteriorates if no tabu list is used. The variable of interest is c*, which stores the cost of the trial solution of minimum cost among all feasible solutions found up to the current instant.

We formulated the cost c as the sum of two terms: $c = c_1 + c_2$. Here, c_1 guides the procedure towards feasible solutions and c_2 reduces the number of seniority violations.

We computed c_1 as follows. In S, let the number of employees assigned to bucket (loc,j) be nemp(loc,j), and let diff(loc,j) = nemp(loc,j) - vacancy(loc,j). Now let

$$c_1 = w_1 * \Sigma \{ diff^2(loc,j) \mid \text{all buckets (loc,j) for which } diff(loc,j) > 0 \}$$

where w_1 is a weight factor, and the summation is over all buckets. Thus only overfull buckets make a contribution to the value of c_1. The weight factor w_1 should be chosen so that the ratio of the initial values of c_1 and c_2 (see below) lies in the range 0.2 to 0.3. For some hard problems a larger or a smaller value of the ratio may be needed to ensure that a feasible solution is found. We computed c_2 as follows. For each employee k, let totviol(k) be the number in S of employees who are junior to k each of whom has caused k a seniority violation. Now take

$$c_2 = w_2 * \Sigma \{ totviol(k) \mid 1 \leq k \leq E \}$$

where the weight factor $w_2 = 2 * E^2 / (nLOC+nJ)$, nLOC being the size of set LOC and nJ the size of set J. The summation is over all employees. The algorithm is not particularly sensitive to the exact term used in the expression for w_2.

.

IV. Experimental Observations

We now summarize our experimental observations. The methods were programmed in C in a Linux environment and run on a Pentium-IV, 1.8 GHz, 1 GB RAM machine. Identical problem instances were run for the two methods. We did not run other methods because they have been already shown to be inferior to SA [3]. We wanted to create random instances of the STP that were realistic and indicative of real life situations. So we kept the number E of employees to be transferred between 500 and 2000. Specialists in Human Resource Management informed us that more than 2000 employees are rarely transferred by any organization in one lot. For all runs we took T = 3. Initially, the E employees to be transferred were randomly assigned to buckets. This information was used for computing chain vacancies. The value of parameter *extra* was chosen to lie between 0% and 40% of E; it was felt that a higher vacancy rate would be unrealistic. The parameter *extra* is calculated as *extra* = MPOST*loc*j*0.5*100/E, where MPOST is a parameter.

TABLE 1 The Staff Transfer Problem
Performance of Simulated Annealing and GSAT

E	LOC, J	POST	Extra (%)	Method	Solved	1st opt	2nd opt	3rd opt	Time (s)
500	15,10	1	15	GSAT	69	209	172	119	12.12
				SA	61	193	173	134	16.10
600	20,10	1	16.6	GSAT	55	243	206	151	16.02
				SA	52	230	205	165	22.99
700	20,10	1	13.7	GSAT	70	289	242	169	24.53
				SA	51	276	237	187	36.68
750	25,10	1	16.6	GSAT	52	303	261	186	12.65
				SA	46	287	262	201	19.17
900	30,10	1	16.6	GSAT	54	362	315	223	42.03
				SA	49	347	309	244	58.17
1000	30,10	1	15	GSAT	47	414	342	244	38.89
				SA	46	390	340	270	80.37
1000	30,10	2	30	GSAT	96	429	343	228	16.32
				SA	92	388	342	270	52.60
1200	40,10	2	33.3	SAT	94	508	413	279	24.35
				SA	86	462	407	331	98.55
1250	40,10	2	32	GSAT	95	532	424	294	27.67
				SA	95	481	422	347	100.99
1500	40,10	2	26.6	GSAT	96	650	514	336	169.72
				SA	88	590	508	402	123.11

Additional vacant positions were created for each value of *extra* and randomly assigned to buckets. For each employee, T transfer options were also randomly created. Transfer options to buckets having no vacancies were not permitted. No restrictions were imposed on which jobs an employee could perform, it being felt that such restrictions were unlikely to affect the runtime. 100 problems were generated for each set. We determined the number of problems solved in a set and the average runtime in seconds. We also computed, per problem instance, the average numbers of employees who

were assigned their first, second and third transfer options. The averages were taken over solved instances. The methods were compared on the basis of three criteria: i) the number of problems solved in each set of 100; ii) the runtime averaged over solved problems, iii) the quality of the solution obtained, quality being determined by the extent to which desirable constraints were satisfied.

Our observations on the experiments are as follows

The Staff Transfer Problem is a difficult problem. The run times are high when E is large. In the earlier implementations (see [2,3]), SA was the best method. But here GSAT(L) generally outperforms SA. Table 1 illustrates the comparative performance of SA and GSAT(L). We see that in most cases GSAT(L) outperforms SA. It solves more instances and runs faster. If we compare the average number of options (1^{st}, 2^{nd} and 3^{rd}) assigned to the employees then also the performance of GSAT(L) is slightly better than that of SA. Of course, not every randomly generated problem instance has a feasible solution. When *extra* = 0, it is quite possible that a problem does not have a feasible solution. From our experimental results it appears that this is rarely the case. Problem instances were all generated randomly. If the transfer scheme described here is used in real life, the transfer options submitted by employees is likely to exhibit bias in favor of certain postings and against certain other postings. This would make the data less random, and the problems could become harder to solve using randomized CSP techniques. One way to resolve this difficulty might be the following. The organization could generate an extra transfer option for each employee, and this could be the option of least preference. If this last option is allocated properly among employees, an acceptable solution would always be found.

V. Conclusion

In this paper, the Staff Transfer Problem is viewed as a CSP. It is converted to a SAT, and experiments indicate that the greedy local search technique GSAT(L) that makes use of tabu search solves problem quite efficiently. In most of the instances it outperforms SA, which was earlier considered the best. But GSAT(L) has a limitation. The number of clauses increases rapidly with increase in the size of the problem. In a machine with 1 GB RAM we could not run instances having more than 1500 employees.

Some interesting issues remain open. The first concerns the formulation of an analytic solution procedure for the STP. If the problem can be suitably expressed in the language of Mathematical Programming, we can compare the runtime and the quality of solution obtained by an analytical procedure with that obtained by GSAT(L). The second relates to the NP-completeness of the Staff Transfer Problem when seniority violations are taken into account. The difficulty level of the problem suggests that it is NP-complete, but this has not yet been proved formally.

References

[1] Acharyya S, *The Satisfiability Problem: A Constraint Satisfaction Approach*, Ph D Thesis, Computer Science & Engineering., University of Calcutta, 2001

[2] Acharyya S, Bagchi A, "Staff Transfers in a Large Organization: A Constraint Satisfaction Approach", *Proc KBCS-98*, International Conference on Knowledge-Based Computer Systems, Mumbai, India, 1998, pp 51-63

[3] Acharyya S, Bagchi A, "Constraint Satisfaction Methods for Solving the Staff Transfer Problem", *OPSEARCH*, vol 42, no 3, September 2005, pp 179-198

[4] Beer M *et al* , *Human Resource Management, A General Manager's Perspective: Text & Cases*, The Free Press, 1985

[5] Dessler G, *Human Resouce Management* (17^{th} Ed), Prentice Hall India, 1997

[6] Fukunaga A S, "Variable Selection Heuristics in Local Search for SAT", *Proc AAAI-97*, American Association for Artificial Intelligence, 1997, pp 275-280

[7] Papadimitriou C H, Steiglitz K, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, 1982

[8] Johnson D S, Aragon C R, McGeoch L A, Schevon C, "Optimization by Simulated Annealing: An Experimental Evaluation, Part II, Graph Coloring and Number Partitioning", *Operations Research*, vol 39, 1991, pp 378-406.

[9] Reeves C R, *Modern Heuristic Techniques for Combinatorial Problems*, Orient Longman, 1993

[10] Mazure B, Sais L, Gregoire E, "Tabu Search for SAT", *Proc AAAI-97*, American Association for Artificial Intelligence, 1997, pp 281-285

[11] Selman B, Kautz H, Cohen B, "Noise Strategies for Improving Local Search", *Proc AAAI-94*, American Association for Artificial Intelligence, 1994, pp 337-343

[12] Selman B, Levesque H J, Mitchell D J, "A New Method for Solving Hard Satisfiability Problems", *Proc AAAI-92*, American Association for Artificial Intelligence, 1992, pp 440-446