Computing 44, 279–303 (1990)

Computing
© by Springer-Verlag 1990

Algorithms for the Maximum Satisfiability Problem*

Pierre Hansen, New Brunswick, U.S.A., and **Brigitte Jaumard,** Québec

Received January 5, 1988; revised June 22, 1989

Abstract — Zusammenfassung

Algorithms for the Maximum Satisfiability Problem. Old and new algorithms for the Maximum Satisfiability problem are studied. We first summarize the different heuristics previously proposed, i.e., the approximation algorithms of Johnson and of Lieberherr for the general Maximum Satisfiability problem, and the heuristics of Lieberherr and Specker, Poljak and Turzik for the Maximum 2–Satisfiability problem. We then consider two recent local search algorithmic schemes, the Simulated Annealing method of Kirkpatrick, Gelatt and Vecchi and the Steepest Ascent Mildest Descent method, and adapt them to the Maximum Satisfiability problem. The resulting algorithms, which avoid being blocked as soon as a local optimum has been found, are shown empirically to be more efficient than the heuristics previously proposed in the literature.

AMS Subject Classifications: 90 Programming, 03 Logic

Key words: Satisfiability, heuristic, computation

Algorithmen für das maximale Erfüllbarkeitsproblem. Es werden bekannte und neue Algorithmen für das maximale Erfüllbarkeitsproblem untersucht. Zunächst geben wir eine Übersicht über verschiedene bisher vorgeschlagene Heuristiken wie z.B. die Approximationsalgorithmen von Johnson und von Lieberherr für das allgemeine maximale Erfüllbarkeitsproblem und die Heuristiken von Lieberherr und Specker, sowie von Poljak und Turzik für das maximale 2–Erfüllbarkeitsproblem. Sodann beachten wir zwei neuere lokale Suchverfahren, wie die Simulated Annealing Methode von Kirkpatrick, Gelatt und Vecchi sowie die Methods des steilsten Anstieges und flachsten Abstieges und adaptieren diese Verfahren für das maximale Erfüllbarkeitsproblem. Es zeigt sich, daß diese Verfahren, die nicht in einem lokalen Optimum stehen bleiben, empirisch effizienter sind als die bisher in der Literatur vorgeschlagenen Heuristiken.

1. Introduction

A central problem in Artificial Intelligence, Logic and Computational Complexity is the Satisfiability problem (SAT) which can be expressed as follows:
Given a collection C of m clauses involving n logical variables $x_1, x_2, \ldots, x_n$, determine whether or not there exists a truth assignment for C such that all clauses are simultaneously satisfied.
SAT has a wide variety of applications such as consistency in expert systems knowl-

* Research of the first author was supported by Air Force Office of Scientific Research Grant #AFOSR-0271 to Rutgers University.
Research of the second author was carried out while visiting the CAIP Center, Rutgers University.

edge bases (see Hayes, Waterman and Lenat [30], Nguyen et al. [43]), integrity constraints in databases (see e.g. Asirelli, de Santis and Martelli [3], Gallaire, Minker and Nicolas [16]).

If the answer to SAT is "no", a natural question, of great practical importance, arises, i.e., how close can one get to satisfiability? Making that question precise leads to the **Maximum Satisfiability problem** (see Karp [38]) studied in this paper. Expressed as a decision problem it can be defined as follows:
Given a collection C of m clauses, involving n logical variables $x_1, x_2, \ldots, x_n$, *and a positive constant k, determine whether or not there exists a truth assignment for C such that at least k clauses can be simultaneously satisfied.*

This problem can also be expressed as an optimization problem and then takes the form:
Given a collection C of m clauses involving n logical variables $x_1, x_2, \ldots, x_n$, *determine a truth assignment for C that satisfies the maximum number of clauses.*

We will consider later a subproblem of MAX–SAT, the **Maximum r–Satisfiability problem** (MAX–rSAT) which can be stated as follows:
Given a collection C of m clauses involving n logical variables $x_1, x_2, \ldots, x_n$ *and such that each clause contains at most r distinct literals, determine a truth assignment for C that satisfies the maximum number of clauses.*

As the MAX–SAT problem is NP-complete, one must often be content with an approximate solution, obtained by some heuristic algorithm. Indeed, the computing time required to find an optimal solution, or to prove the optimality of such a solution when it is found, becomes prohibitive when the problem gets large.

Ascent methods constitute an important class of heuristics. In these methods, an initial solution is found and subjected to a sequence of local changes which improve each time the value of an objective function until a local optimum is found. The objective function may be given in the problem formulation or may be an evaluation function expressing the difference between the current solution and a target solution. The main drawback of ascent algorithms is that they are unable to get out of a local optimum. For some rare classes of problems all local optima are also global optima and ascent methods, prominent among which are the so-called greedy algorithms, yield optimal solutions. Much more often, there are many local optima and the first one found may be far from globally optimal.

Recently, local improvement heuristic methods have been proposed which include features allowing to get out of local optima. Exploiting an analogy with a problem of thermodynamical statistics, Kirpatrick, Gelatt and Vecchi [39] have proposed a general approach to combinatorial optimization called **Simulated Annealing**. Following this approach, an initial solution is found and evaluated. Then a local change is randomly obtained and its effect on the value of the objective function is assessed. If there is an improvement, the local change is accepted. If there is some deterioration, a probability of acceptance is calculated. This probability decreases as the amount of deterioration and the elapsed time (or the number of local changes already considered) increase. A random number in $[0, 1]$ is generated and the local change is accepted if and only if it is smaller than the probability just calculated. The procedure stops as soon as no local change has been accepted during a sequence of a given number of iterations. Otherwise another local change is considered.

This new approach has been widely studied and applied. It has proved to be competitive and sometimes superior to the best heuristics known for some classes of problems (e.g. quadratic assignment problems, see Burkard and Rendl [6]) but not all of those to which it has been applied.

An alternate approach, also allowing to get out of a local optimum has been proposed by one of the authors [26] under the name of **Steepest Ascent Mildest Descent**. Following this approach, an initial solution is found and local changes in the direction of steepest ascent are performed until a local optimum is reached. When this is the case the corresponding solution together with its value are noted, if it is the best one found so far. Then a local change is done along the direction of mildest descent and a reverse change is forbidden for a given number of iterations. The procedure terminates if no improvement in the value of the best found solution is observed during a cycle of iterations of fixed length. A similar algorithmic scheme was independently proposed by Glover [19] under the name of Tabu Search.

A comparison of both heuristics, as applied to the maximization of quadratic 0–1 functions (see El Baamrani [14], Hansen et al. [29]) shows that the solutions obtained by Steepest Ascent Mildest Descent are on average better than those obtained by Simulated Annealing and the computing times lower. Experiments for quadratic assignment problems (see Stephany [47]) lead to similar conclusions for problems with less than thirty facilities; for larger quadratic assignment problems computing times with the Steepest Ascent Mildest Descent heuristic tend to be prohibitive. Both heuristics are applied in this paper to the Maximum Satisfiability problem and comparisons are made with some previous heuristics. Many further applications of Steepest Ascent Mildest Descent or Tabu Search have been done recently by various researchers. Surveys are given by Glover [20] and Hertz and de Werra [32].

The paper is organized as follows. The Maximum Satisfiability problem and its different mathematical formulations are presented in the next section. Some applications of the MAX–SAT problem are described in the last paragraph of that section. The heuristics previously proposed for the Maximum Satisfiability problem are presented, with some simplifications and a uniform notation, in Section 3: first those for the general case (Johnson [36], Lieberherr [40]), and then those which are specific to MAX–2SAT (Lieberherr and Specker [41], Poljak and Turzik [44]). Section 4 is devoted to new approaches with Ascent Descent methods; precise formulations of Simulated Annealing and of Steepest Ascent Mildest Descent are given in subsections 4.1 and 4.2 respectively, and their specializations to MAX–SAT are studied in subsection 4.3. Computational experiments are reported on in Section 5. An analysis of the performance of Ascent Descent methods is presented in subsection 5.1 and a comparison with the other heuristics (and with an exact algorithm in the case of MAX–2SAT) is made in subsection 5.2. Conclusions are drawn in Section 6.

2. Formulations of the Maximum Satisfiability Problem

2.1. Definition of the Problem

Let $U = \{u_1, u_2, \ldots, u_n\}$ be a set of boolean variables. A **truth assignment** for U is a function t: $U \rightarrow \{\text{True}, \text{False}\}$. If $t(u) = \text{True}$, we say that u is true under t; if $t(u) = \text{False}$ we say that u is false under t. If u is a variable in U, then u and $\bar{u}$ are literals over U. The literal u is true under t if and only if the variable u is true under t; the literal $\bar{u}$ is true if and only if the variable u is false.

A **clause** over U is a disjunction of literals. It is satisfied by a truth assignment if and only if at least one of its literals is true under that assignment. A **formula** is a conjunction C of clauses and is satisfiable if and only if there exists some truth assignment for U that simultaneously satisfies all the clauses in C. Such a truth assignment is called a **satisfying truth assignment** for C. The **Maximum Satisfiability** problem is defined as follows:
Instance: *Set U of variables, collection C of clauses, positive integer $k \leq |C|$.*
Question: *Is there a truth assignment for U that simultaneously satisfies at least k of the clauses in C?*
MAX–SAT is NP-complete even in the particular case when there are two variables per clause (Garey, Johnson and Stockmeyer [18]), or when each clause involves at most one positive literal, i.e., is of the Horn type (Jaumard and Simeone [35]). However when each variable has at most two occurrences (the variable being complemented or not in one or both of its occurrences), the problem is reducible to a minimum cost network flow problem (Jaumard [34]), and hence is polynomial.

MAX–SAT contains the Satisfiability problem (SAT) which may be expressed as follows:
Instance: *Set U of variables, collection C of clauses over U.*
Question: *Is there a satisfying truth assignment for C?*
The latter problem has two well known polynomial-time instances: 2–Satisfiability (2-SAT) where each clause contains at most two literals and Horn satisfiability (HORN-SAT) where each clause is of Horn type. The former problem can be solved in $O(|C|)$ time by an algorithm of Aspvall, Plass and Tarjan [4] (see also Even, Itai and Shamir [15]) and the latter one can be solved in $O(l)$ time (where l is the size of the problem, i.e., the total number of occurrences of the variables) by an algorithm of Dowling and Gallier [10] (see also Henschen and Wos [31], Jones and Laaser [37]). Other polynomially solvable cases are described in Arvind and Biswas [2], Dubois [11], Tovey [49], Yamasaki and Doshita [50]. When there are three or more literals by clause, SAT is NP-complete (Cook [8]).

2.2. Mathematical Formulations

We are now interested in the optimization form of the MAX–SAT problem: given a set $C = \{C_1, C_2, \ldots, C_m\}$ of clauses over a set $U = \{u_1, u_2, \ldots, u_n\}$ of boolean variables, determine a truth assignment which satisfies the maximum number of clauses of C. For any literal u^{α}, let u^{α} with $\alpha \in \{0,1\}$ equal u if $\alpha = 1$ and $\bar{u}$ if $\alpha = 0$. A clause C_j of C can be expressed as a disjunction of literals $C_j = \bigvee_{i \in I_j} u_i^{\alpha_{ij}}$ where I_j is the subset of $\{1, 2, \ldots, n\}$ corresponding to the indices of the variables in C_j.

2.2.1. Linear 0–1 Programming

We associate a 0–1 variable x_i to each variable u_i of U such that 0 corresponds to the value *False* and 1 to the value *True*. Moreover we associate to each clause C_j a 0–1 variable y_j. Then MAX–SAT can be formulated as follows:

$$\begin{aligned} &\text{Maximize } y = \sum_{j=1}^{m} y_j \\ &\text{Subject to: } \quad \sum_{i \in I_j} x_i^{\alpha_{ij}} \geq y_j \qquad j = 1, 2, \ldots, m \\ &\qquad\qquad\qquad x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n \\ &\qquad\qquad\qquad y_j \in \{0,1\} \qquad j = 1, 2, \ldots, m. \end{aligned}$$

As the sum of the y_j is maximized and as each y_j appears as the right-hand side of one constraint only, y_j will be equal to 1 if C_j is true and equal to 0 if C_j is false. Deleting the objective function and setting all variables y_j at 1 yields a formulation of the SAT problem studied by Blair, Jeroslow and Lowe [5].

The linear 0–1 programming formulation of MAX–SAT suggests that this problem could be solved by a branch-and-bound method using linear programming relaxation, i.e., replacing the constraints $x_i \in \{0,1\}$ and $y_j \in \{0,1\}$ by $x_i \in [0,1]$ and $y_j \in [0,1]$. Unfortunately this is not likely to work well in practice because the solution $x_i = \frac{1}{2}$ $(i = 1, 2, \ldots, n)$, $y_j = 1$ $(j = 1, 2, \ldots, m)$ is feasible for the linear programming relaxation unless there exists some constraints containing only one variable. The bounds so obtained would be very poor. Better bounds might be obtained using Chvatal cuts, as done by Hooker [33] for the problem SAT.

2.2.2. Unconstrained Nonlinear 0–1 Programming (Pseudo-Boolean Programming)

Let $\bar{\alpha}_i = 1 - \alpha_i$. Then the negation of clause $C_j = \bigvee_{i \in I_j} u_i^{\alpha_{ij}}$ is the term $\bar{C} = \bigwedge_{i \in I_j} u_i^{\bar{\alpha}_{ij}}$; it is true if and only if no literal $u_i^{\alpha_{ij}}$ of C_j is true. Solving MAX–SAT is equivalent to finding the truth assignment such that the smallest number of clauses $\bar{C}_j$ are true. This in turn is equivalent to the following unconstrained nonlinear 0–1 minimization problem, or unconstrained pseudo-boolean minimization problem (see Hammer and Rudeanu [22]):

$$\begin{aligned} &\text{Minimize } z = \sum_{j=1}^{m} \prod_{i \in I_j} x_i^{\bar{\alpha}_{ij}} \\ &\qquad\qquad x_i \in \{0,1\} \qquad i = 1, 2, \ldots, n \end{aligned}$$

This formulation has recently been used by Sutter [48] in a branch-and-bound algorithm for MAX–2SAT.

2.3. Applications

2.3.1. Inconsistency in Expert-Systems Databases

In expert-systems (see e.g. Hayes, Waterman and Lenat [30], Nguyen et al. [43]), databases are used in which the information about objects under study is usually expressed by implications. Well-known methods of boolean algebra allow to rewrite these implications as a set of clauses. An important practical problem is to check the consistency of the database after updating by adding and deleting some of the implications. If the database is not consistent, it is desirable to find the smallest set of implications which should be deleted, or all smallest such sets of implications, in order to restore consistency. Such a set can be obtained by solving the MAX–SAT problem for the clauses corresponding to all implications in the database.

2.3.2. Consistency of Database Queries

Problems similar to the above one (see e.g. Gallaire, Minker and Nicolas [16] and Asirelli, de Santis and Martelli [3]) arise in evolutive databases which use logic programming tools in order to store informations. Typically, in the more general problem of integrity constraint, one wants to know the reason for which the system cannot answer some queries. One possible reason is lack of logical consistency of the set of facts contained in the database. Due to the large number of facts in current databases and also to the numerous dynamic updates, this problem is hard to solve in practice; so efficient heuristics are needed.

3. Previous Heuristics

3.1. General Maximum Satisfiability Problem

3.1.1. Johnson [1974]

Johnson proposed two approximation algorithms that can be implemented in time $O(\ell . \log(\ell))$ where ℓ denotes the size of the problem. Each heuristic is of greedy type and works by generating a truth assignment TRUE. The first heuristic, JOHN1, is presented below.

Heuristic procedure JOHN1.

```
1. TRUE $\leftarrow \phi$; LEFT $\leftarrow C$; LIT $\leftarrow U \cup \bar{U}$; $z$(TRUE) $\leftarrow 0$;
2. If LEFT $= \phi$ *then stop and return* $z$(TRUE) *as the approximate optimum*;
```

```
3. Let $u^\alpha$ be the literal in LIT which is contained in the most clauses of LEFT and let
   UT be the set of clauses in LEFT which contain $u^\alpha$;
4. Set: $z(\text{TRUE}) \leftarrow z(\text{TRUE}) + |\text{UT}|$;
        $\text{LEFT} \leftarrow \text{LEFT} \setminus \text{UT}$;
        $\text{TRUE} \leftarrow \text{TRUE} \cup \{u^\alpha\}$;
        $\text{LIT} \leftarrow \text{LIT} \setminus \{u, \bar{u}\}$;
5. Goto 2.
```

Let z^* denote the maximum number of satisfied clauses and q the minimum number of literals contained in any clause of the problem. Johnson showed that for all $q \geq 1$ and for any approximate solution TRUE obtained by the JOHN1 heuristic:

$$z(\text{TRUE}) \geq \frac{q}{q+1} z^*$$

and the bound is sharp for all sufficiently large ℓ.

The second heuristic, JOHN2, takes into account "wounded" clauses, i.e., clauses containing literals to which the value false has been assigned as the truth assignment is generated.

Heuristic procedure JOHN2.

```
1. Assign to each clause $C_i \in C$ a weight $w(C_i) = 2^{-|C_i|}$;
     Set: $\text{TRUE} \leftarrow \phi$;
          $\text{LEFT} \leftarrow C$;
          $\text{LIT} \leftarrow U \cup \bar{U}$;
          $z(\text{TRUE}) \leftarrow 0$;
2. If $\text{LEFT} = \phi$ then stop and return $z(\text{TRUE})$ as the approximate optimum;
3. Let $u$ be any literal occurring in both LIT and a clause of LEFT;
   Let UT be the set of clauses in LEFT containing $u$ and UF be the set of clauses in LEFT
   containing $\bar{u}$;
4. If $\sum_{C_i \in \text{UT}} w(C_i) \geq \sum_{C_i \in \text{UF}} w(C_i)$
     then $\text{TRUE} \leftarrow \text{TRUE} \cup \{u\}$;
       $z(\text{TRUE}) \leftarrow z(\text{TRUE}) + |\text{UT}|$;
       $\text{LEFT} \leftarrow \text{LEFT} \setminus \text{UT}$;
       For each $C_i \in \text{UF}$ do $w(C_i) \leftarrow 2 \cdot w(C_i)$;
     else $\text{TRUE} \leftarrow \text{TRUE} \cup \{\bar{u}\}$;
       $z(\text{TRUE}) \leftarrow z(\text{TRUE}) + |\text{UF}|$;
       $\text{LEFT} \leftarrow \text{LEFT} \setminus \text{UF}$;
       For each $C_i \in \text{UF}$ do $u(C_i) \leftarrow 2 \cdot w(C_i)$;
   Endif;
5. $\text{LIT} \leftarrow \text{LIT} \setminus \{u, \bar{u}\}$; goto 2.
```

The worst case bound of the JOHN2 heuristic is much better than that of the JOHN1 heuristic. For all $q \geq 1$ and for any approximate solution $z(\text{TRUE})$ obtained by the JOHN2 heuristic:

$$z(\text{TRUE}) \geq \frac{2^q - 1}{2^q} z^*,$$

and the bound is sharp for all sufficiently large ℓ.

Note that since the choice of which literal to consider at Step 3 in JOHN2 is left largely undetermined, there is some potential for improvements which might affect its average case behavior. Furthermore, one may notice that both algorithms do not necessarily stop at a local optimum, i.e., a truth assignment such that changing the truth value of a single variable does not improve the value of the objective function. To illustrate this, consider the following example:

$$C = \{u_1 \vee u_2, u_1 \vee \bar{u}_2, \bar{u}_1 \vee u_2, \bar{u}_1 \vee \bar{u}_2, \bar{u}_1 \vee u_3, \bar{u}_1 \vee \bar{u}_3, u_2 \vee \bar{u}_3,$$
$$u_2 \vee u_4, \bar{u}_2 \vee u_3, \bar{u}_2 \vee \bar{u}_3, u_3 \vee u_4, \bar{u}_3 \vee \bar{u}_4\}.$$

Assume literals are ranked by increasing indices, the uncomplemented one preceding the complemented one each time. When there are ties in JOHN1 and in JOHN2, we use the lexicographic order. Applying the JOHN1 heuristic, we successively select the literals $\bar{u}_1$, u_2, u_3 and u_4 to be set to true, which yields a solution vector $u^* = (\text{F}, \text{T}, \text{T}, \text{T})$ of value 9 (where T stands for true and F for false). For the JOHN2 heuristic, let us consider the criterion of selecting the literal u with the largest sum of weights $w(C_i)$ for all clauses C_i to which u belongs. Applying the JOHN2 heuristic with this criterion, or only with the lexicographic order yields again the solution $u^* = (\text{F}, \text{T}, \text{T}, \text{T})$. But u^* is not a local optimum since the value of the vector $(\text{F}, \text{T}, \text{T}, \text{F})$ is 10.

3.1.2. Lieberherr [1982]

Lieberherr [40] proposed an efficient approximation algorithm ψ-MAXMEAN for the generalized maximum ψ-satisfiability problem which includes MAX–SAT. Moreover, he showed that, form an extremal point of view his algorithm is the best possible in the class of polynomial algorithms. The generalized maximum ψ-satisfiability problem is defined as follows (Schaefer [46]): Let $\psi = \{R_1, R_2, \ldots, R_m\}$ be any finite set of logical relations. A logical relation is defined to be any subset of $\{0, 1\}^r$ for some integer $r \geq 1$, and a ψ-formula is any conjunction $R_1 \wedge R_2 \wedge \cdots \wedge R_m$ of such relations. The ψ-satisfiability problem is the problem of deciding whether a given ψ-formula is satisfiable. The maximum ψ-satisfiability problem is defined by: given a ψ-formula F, find a 0–1 assignment to the variables of F which satisfies the maximum number of relations (clauses in the case of MAX–SAT).

Let us recall one more definition before stating the main results of Lieberherr, as well as his algorithm ψ-MAXMEAN. A ψ-formula F is called **symmetric** if any permutation of the variables in the formula returns the same formula up to a permutation of the relations.

Let ψ be a set of relations, Γ a set of symmetric ψ-formulas and F a ψ-formula of Γ with n variables. Let $mean_k(F)$ denote the average number of satisfied clauses among all assignments which set exactly k variables to 1 and $\text{maxmean}(F) = \max_{0 \leq k \leq n} \text{mean}_k(F)$.

a) If F is symmetric then there is a polynomial algorithm MAXMEAN which satisfies the fraction:

$$r_\Gamma = \inf_{F \in \Gamma} \max_{\text{all } 0-1 \text{ assignments } J \text{ of } F} \frac{SA(F,J)}{rl(F)}$$

of the relations for a given ψ-formula $F \in \Gamma$, where $SA(F,J)$ is the number of satisfied relations in formula F under assignment J, and $rl(F)$ the number of relations in the ψ-formula F.

b) If the ψ-satisfiability problem for the formulas in Γ is NP-complete and Γ is closed under concatenation of formulas with disjoint variables, then: for any rational $r' > r_\Gamma$ and ψ-formulas in Γ finding a truth assignment which satisfies the fraction r' of the clauses is NP-complete.

c) Algorithm MAXMEAN satisfies at least maxmean(F) relations for $F \in \Gamma$.

d) If Γ is closed under renaming, then the problem of finding an assignment that satisfies strictly more than maxmean(F) relations for $F \in \Gamma$, is NP-equivalent.

Given a ψ-formula F, the MAXMEAN algorithm outputs a 0–1 assignment which satisfies at least maxmean(F) relations and can be stated as follows:

Procedure MAXMEAN

1. *Find* k $(0 \leq k \leq n)$, *so that* $maxmean(F) = mean_k(F)$ *by a linear search.*
2. *Apply algorithm* MEAN *to* F *and* k *to find an assignment satisfying at least* $maxmean(F)$ *relations.*

Given a ψ-formula and an integer $k(0 \leq k \leq n)$, the algorithm MEAN outputs an assignment which satisfies at least $mean_k(F)$ clauses (the number of variables to which 1 is assigned might be $< k$) and can be stated as follows:

Procedure MEAN

```
For all variables $x_i$ in $F$ do
  if $mean_{k-1}(F_{/x_i=1}) > mean_k(F_{/x_i=1})$
    then $x_i \leftarrow 1$; $k \leftarrow k - 1$; $F \leftarrow F_{/x_i=1}$
    else $x_i \leftarrow 0$; $F \leftarrow F_{/x_i=0}$
  endif
Endfor.
```
($mean_{-1}(F)$ *is defined to be zero.*)

3.2. Maximum 2–Satisfiability

3.2.1. Lieberherr and Specker [1981]

Independently of the algorithm MAXMEAN for the general case, Lieberherr and Specker [41] proposed another algorithm for MAX–2SAT. They first show that every

2–satisfiable formula has a truth assignment that satisfies at least the fraction h of its clauses, where $h = \frac{\sqrt{5}-1}{2} \sim 0.618$ (the reciprocal of the "golden ratio"). Then they provide a polynomial-time algorithm that will find for any 2–satisfiable formula a truth assignment satisfying at least the fraction h of its clauses. Furthermore they prove that, for any rational $h' > h$, finding a truth assignment satisfying at least the fraction h' of the clauses of any 2–satisfiable formula is NP-complete.

The algorithm ENUM of Lieberherr and Specker is presented below:

Procedure ENUM

1. *For each variable x_i which occurs complemented in a clause of length* 1, *replace all occurrences of the literals x_i and $\bar{x}_i$ by their complements (afterwards the clauses of length* 1 *contain only positive literals.)*
2. *Compute the first prime p greater than or equal to n, and let W be a set of p variables containing all those from X.*
3. *Enumerate the set $I(n)$ of truth assignments (or interpretations):*

$$I(n) = \bigcup_{k=0}^{p} \bigcup_{q=1}^{p-1} \bigcup_{r=1}^{p} \text{INT}(n, k, q, r)$$

where the truth assignments $\text{INT}(n, k, q, r)$ *are defined in the following way. Let $RP[k]$ be an arbitrary permutation of p variables. Then the variable x_i in $W(1 \leq i \leq n)$ is set to* 1 *by the truth assignment* $\text{INT}(n, k, q, r)$ *if and only if*

$$((q \cdot RP[k](i) + r) \bmod p) \leq k,$$

otherwise x_i is set to 0.
Keep the truth assignment which satisfies the maximal number of clauses in F.

The ENUM algorithm is a polynomial algorithm with an overall running time $O(n^3\ell)$ where ℓ is the total number of occurrences of literals in the formula F. It is easy to build an example (cf Lieberherr and Specker [41]) showing that the heuristic JOHN2 will not in general provide a truth assignment satisfying at least a fraction h of the clauses. Using the ENUM algorithm, there is no guarantee that the best truth assignment corresponds to a local optimum.

3.2.2. Poljak and Turzik [1982]

Recall that a bipartite subgraph of a graph $G = (V, E)$ is such that the vertices can be partitioned into two subsets V_1 and V_2 and all edges have one extremity in V_1 and the other in V_2. Edwards [12, 13] provides an estimate for the maximum number of edges of a bipartite subgraph. From this estimate, Poljak and Turzik [44] deduce an extremal bound for the Maximum 2–Satisfiability problem and define a polynomial algorithm providing a truth assignment which allows to reach this bound.

Given a quadratic formula, they define a graph $G = (V, E)$ as follows: each variable $u_i \in U$ is associated with a vertex u_i and each clause $u_i^{\alpha_i} \vee u_j^{\alpha_j}$ such that $i \neq j$ is associated with an edge (u_i, u_j). In most cases, G is a multigraph, i.e., has parallel edges.

Poljak and Turzik prove that one can always find a truth assignment which satisfies at least $\frac{3m - p}{4}$ clauses of the quadratic formula where p denotes the number of linear clauses. Moreover they provide an $O(m)$ linear-time algorithm (POLTUR) to obtain such a truth assignment and we recall it below. They show that when the graph G is simple, i.e., the quadratic formula does not contain any pair of clauses involving the same variables, the bound can be improved to $\frac{3m - p}{4} + \frac{1}{4}\left[\frac{1}{2}(n - k)\right]$ clauses where k denotes the number of components of G. They also provide an $O(n^3)$ polynomial-time algorithm to obtain such an assignment. The reader is referred to their paper for further details on the latter algorithm.

Procedure POLTUR heuristic

```
1. TRUE $\leftarrow \phi$; VAR $\leftarrow U$; LEFT $\leftarrow C$;

2. If *no variable in U is involved in any clause of* LEFT
      then TRUE1 $\leftarrow$ TRUE1 $\cup U$;
           TRUE2 $\leftarrow U \cup \bar{U} \setminus$ TRUE1;
           *evaluate* z(TRUE1) *and* z(TRUE2);
           *return* z(TRUE) = *max* (z(TRUE1), z(TRUE2)) *as the approximate solution*;
           *stop*;
   Endif;

3. *Let* $u_i$ *be any variable occurring in both U and a clause of* LEFT;
   *Set*: VAR $\leftarrow$ VAR$\setminus\{u_i\}$;
   *Let* CT *be the set of clauses of* LEFT *involving only variables of* $U\setminus$VAR *and* CT1 *be*
   *a subset of clauses of* CT *which contain both a positive and a negative literal*;
   *Set*: LEFT $\leftarrow$ LEFT$\setminus$CT; $\delta \leftarrow 0$;

4. For *every clause* $C(\alpha_i, \alpha_j) = u_i^{\alpha_i} \vee u_j^{\alpha_j}$ *of* CT do
      If $u_j \in$ TRUE then
        If $C(\alpha_i, \alpha_j) \in$ CT1 then $\delta \leftarrow \delta + 1$ else $\delta \leftarrow \delta - 1$ endif
           else
        If $C(\alpha_i, \alpha_j) \in$ CT1 then $\delta \leftarrow \delta - 1$ else $\delta \leftarrow \delta + 1$ endif
      Endif
   Endfor;
      If $\delta \geq 0$ then TRUE $\leftarrow$ TRUE $\cup \{u_i\}$
                else TRUE $\leftarrow$ TRUE $\cup \{\bar{u}_i\}$
   Endif;

5. *Goto* 2.
```

It follows from the estimate $\frac{3m - p}{4}$ that for any approximate solution TRUE obtained by the POLTUR heuristic

$$z(\text{TRUE}) \geq \frac{3}{4}z^* - \frac{p}{4}.$$

For MAX − 2SAT the worst case bounds for JOHN1 and JOHN2 are respectively equal to $\frac{2}{3}z^*$ and $\frac{3}{4}z^*$ if there are unit clauses, and both equal to $\frac{z^*}{2}$ otherwise. If we consider a quadratic formula with linear clauses, i.e., $p \neq 0$, since z^* is always greater than p, the worst case analysis gives advantage to the POLTUR heuristic rather than to the JOHN2 (or JOHN1) heuristic. However if the quadratic formula does not contain any unit clause, the worst case bound of POLTUR is equivalent to the one of JOHN2, hence better than the one of JOHN1.

As in the JOHN2 heuristic, the choice of which literal to consider at Step 3 is left undetermined, which might lead to some improvement in the average case behavior. Again one can notice that the POLTUR heuristic does not necessarily stop at a local optimum, as follows. Let us consider the same set of clauses C as in the paragraph 3.1.1. Assume that we select the literals in the order u_1, u_2, u_3 and u_4. We obtain TRUE $= \{u_2, u_4\}$. The values of the vectors (F, T, F, T) and (T, F, T, F) are respectively 10 and 8 so that the optimum vector is $u^* =$ (F, T, F, T). But u^* is not a local optimum since the value of (F, F, F, T) is 11.

4. Ascent Descent Methods

4.1. Simulated Annealing

In the early days of computer science, Metropolis et al. [42] proposed a heuristic to simulate the behavior of a collection of atoms initially in equilibrium at temperature t and subjected to cooling. It was well known that rapid cooling would block the system in a high energy state corresponding to disorder whereas slow cooling, or annealing, would bring the system into an ordered, low energy state. This behavior was reproduced by the proposed simulation program.

The procedure first considers a system of atoms in motion with energy E and temperature t. An atom is randomly chosen and a random displacement applied to it. The implied change in energy dE of the system is calculated. If dE is negative then the displacement is accepted and E reduced by dE. If dE is positive, the probability for the change dE to take place is evaluated, following Boltzmann's law: $P(dE) = \exp\left(-\frac{dE}{kt}\right)$ where k is Boltzmann's constant and the temperature t is evaluated on the Kelvin scale. Then the probability $P(dE)$ is compared with the value of a random variable x uniformly distributed in $[0, 1]$. The displacement is accepted if and only if x is less than $P(dE)$. This basic iteration is repeated many times and as predicted by Boltzmann's law, the system evolves to the most likely state at temperature t. Then t is multiplied by a constant a in $[0, 1]$ and the sequence of iterations repeated. The minimal value of the energy decreases with t as the probability to accept a displacement which would increase E diminishes. When t is sufficiently low, no more random displacements are accepted. The system is "frozen" in a quasi stable state corresponding to a local minimum of E.

Thirty years after the publication of Metropolis et al.'s method, Kirpatrick, Gelatt and Vecchi [39] proposed to apply a smilar approach to the optimization of combinatorial problems. Independently, Cerny [7] considered the use of such a method to solve the traveling salesman problem.

This new approach to heuristics for combinatorial optimization has the following advantages:

1. The heuristic does not stop at the first local optimum found but, with some probability, proceeds to further exploration of the solution set.
2. According to Kirkpatrick et al., the heuristic exhibits some sort of natural "divide and conquer" behavior at high temperature, i.e., the main features of the optimal solution become apparent and, to some extent, it breaks up into sub-problems.
3. The theory of thermodynamical statistics can be brought to bear to exhibit some possible properties of the solution set, such as the existence of many local optima with values close to the global optimum.
4. Under reasonable assumptions the heuristic converges (perhaps very slowly), to a global optimum (see e.g. Anily and Federgruen [1]).

We now present a pidgin-algol program for the Simulated Annealing approach following Burkard and Rendl [6]. Suitable changes are introduced in order to consider here the case of maximization.

Heuristic procedure SA

```
Logical change
Real, a
Set t to an initial high value;
Select a feasible solution S and calculate the objective value $z(S)$;
Change ← .true.;
While change do
  Begin
    Change ← .false.;
    Repeat rep times
      Begin
        Transform S into S' by a random move in the neighbourhood of S and
        calculate $\delta$ the corresponding change in the objective function value;
        If $\delta \geq 0$ then goto accept;
        $P(\delta) \leftarrow \exp(\delta/t)$;
        Generate a random value for a variable x uniformly distributed in [0, 1];
        If $x < P(\delta)$ then goto accept else goto exit;
        Accept: $S \leftarrow S'$;
                $z(S) \leftarrow z(S) + \delta$;
                If $\delta \neq 0$ then change ← true;
        Exit;
    End;
    $t \leftarrow t.a$;
End.
```

The logical variable *change* is used to check if no change in the objective function occurs during a cycle of *rep* random trials. The values of the parameters t, *rep* and a have a strong influence on the performance of the heuristic. Finding the best values for a class of problems usually requires a long and systematic experimental study. *Rep* is often taken to be a low order power of the problem size, or some surrogate such as the number of variables. The value t must be chosen sufficiently high for most or all possible changes to be acceptable during the first stage of the resolution. In other words, $P(\delta)$ should initially be close to 1. The value a must be chosen sufficiently large in order to allow a slow cooling and avoid premature freezing in a low value local maximum. As it strongly influences computing time a should not be too large either.

4.2. Steepest Ascent Mildest Descent

The Simulated Annealing heuristic proceeds by local search and is not necessarily blocked as soon as a new local optimum is found. Nevertheless, the direction of the local changes are little exploited. If the objective function decreases, the change is always accepted however small it may be; if the objective function increases, the change is also accepted with some probability. It therefore seems desirable to devise a method which expoits information on the direction of steepest descent and yet retains the property of not being blocked at the first local optimum found. The Steepest Descent Mildest Ascent approach constitute such a method. Local changes in the direction of steepest descent are performed until a local optimum is obtained. Then a local change along a direction of mildest ascent takes place and the reverse move is forbidden for a given number of iterations to avoid cycling with a high probability. Further specialized devices for detecting and breaking cycling can be added and will be outlined after the presentation of the general approach.

We now propose a pidgin-algol program for the Steepest Descent Mildest Ascent heuristic similar to that one for Simulated Annealing presented in the last section.

Heuristic procedure SAMD

```
Logical change
Select a feasible solution $S$ and calculate the objective value $z(S)$;
Set $f_j = 0$ for $j \in J$, index set of the directions of change;
Change $\leftarrow$ .true.;
$S^* \leftarrow S$;
$z^* \leftarrow z(S)$;
While change do
  Begin
    Change $\leftarrow$ .false.;
    Repeat rep times
      Begin
        Transform S into the solution $S_j$ modified in the $j^{th}$ direction and evaluate
          $\delta_j = z(S_j) - z(S)$ for $j \in J$ with $f_j = 0$;
        Let $\delta_k = \max\{\delta_j / j \in J, f_j = 0\}$;
        $S \leftarrow S_k$;
        If $\delta_k \leq 0$ then $f_k \leftarrow p$;
        If $z(S_k) > z^*$ then $z^* \leftarrow z(S_k)$;
                               $S^* \leftarrow S_k$;
                               Change $\leftarrow$ .true.;
        $f_j \leftarrow f_j - 1$ for all $j \in J$ such that $f_j > 0$;
      End;
  End.
```

The logical variable *change* is used to check if a better solution than previously known is found during a cycle of *rep* local changes. The best solution found so far is kept in S^* and its value in z^*. The parameters f_j denote the remaining number of iterations during which a local change in direction j is forbidden. When a change in a descent direction j is done, f_j is set at the value p. The larger the value of p the more difficult it will be to come back to a local optimum, but also to explore its vicinity.

While forbidding local reverse changes, as just described, guarantees to get out of a local optimum, it does not entirely prohibit the possibility of cycling. The following device called "local optima checking" allows to check for and break cycles. Each time a local optimum S^k is found for the first time, it is introduced in a list and a counter d^k for directions used is set equal to 1. When getting to a local optimum, it is first checked whether it belongs to the list of local optima already found (this can be done easily, e.g. by using hash coding). If this is the case, the counter d^k of the directions used is augmented by one and a local change is performed along the corresponding d^k-th mildest descent direction. Otherwise the first direction is used as described above. If, for some solution, all directions have been explored, the procedure stops. Alternatively, all directions could be considered as admissible again after an increase of the parameter p.

4.3. *Application of the Ascent Descent Methods*

4.3.1. Local Changes and Initial Solution

In any ascent-descent method, the set of local changes should be such that any solution can be obtained from any other one after an adequate sequence of such changes. In the MAX–SAT problem one may consider an initial solution which is randomly generated, i.e., a vector of values equal to true or false, and as local changes the complementation of one of these variables, i.e., exchange from true to false or from false to true. Alternatively, one could take as an initial vector a solution given by some heuristic (see Section 3).

4.3.2. Evaluation Functions

In order to evaluate the effect of a local change, different factors can be taken into account such as the number and/or the cardinality of unsatisfied clauses which become satisfied, the number and/or the cardinality of satisfied clauses which become unsatisfied, the number and/or the cardinality of clauses containing the switching literal and which remain satisfied, the difference between the number of occurrences of the variable in direct and in complemented form, and so on.

A simple evaluation function, denoted by f_0, is the following:
Let UNSAT$_k$ (resp. SAT$_k$) be the set of unsatisfied (resp. satisfied) clauses involving the literal u_k or $\bar{u}_k$, and which become satisfied (resp. unsatisfied) when switching the value of u_k. Define the switching variable as the one for which:

$$|\text{UNSAT}_k| - |\text{SAT}_k|$$

is maximum.

Other evaluation functions are discussed in the original version of this paper [27].

4.3.3 Quality of Local Optima

Let z^* denote the optimum value and q the minimum number of literals contained in a given clause of the problem. Applying the Steepest Ascent Mildest Descent method with the evaluation function f_0 leads, before descent takes place, to a local maximum, i.e., to a solution such that no increase in the number of satisfied clauses is obtained by switching the value of any single variable. Such solutions have the same worst case bound as the heuristic JOHN1, as we now show.

Theorem 1: *Any local optimum of any instance of* MAXSAT *with at least q literals per clause is such that:*

$$z \geq \frac{q}{q+1} m,$$

and the bound is sharp.

Proof. Assume the switch defined by the evaluation function f_0 does not allow any new increase of the number of satisfied clauses. Then for any literal u_k: $|\text{UNSAT}_k| \leq |\text{SAT}_k|$.

Let UNSAT (resp. SAT) denote the set of unsatisfied (resp. satisfied) clauses. From the previous relation:

$$q|\text{UNSAT}| \leq \sum_k |\text{UNSAT}_k| \leq \sum_k |\text{SAT}_k| \leq |\text{SAT}| = z,$$

and, as $m = |\text{UNSAT}| + |\text{SAT}|$, $z \geq \dfrac{q}{q+1} m$.

To show that the bound is sharp consider the following example:

$C=\{\bar{u}_1 \vee u_2, \bar{u}_2 \vee u_3, \bar{u}_3 \vee u_1, \bar{u}_4 \vee u_5, \bar{u}_5 \vee u_6, \bar{u}_6 \vee u_4, \bar{u}_1 \vee \bar{u}_4, \bar{u}_2 \vee \bar{u}_5, \bar{u}_3 \vee \bar{u}_6\}$.

The solution $U = \{\text{T}, \text{T}, \text{T}, \text{T}, \text{T}, \text{T}\}$ is locally optimal, with $z = 6$ and $m\dfrac{q}{q+1} = 6$. ■

Corollary 2: $z \geq \dfrac{q}{q+1} z^*$.

Proof. $m \geq z^*$. ■

We now focus on the maximum number r of literals per clause. Consider MAX–rSAT and let p_i denote the number of clauses with i literals, $i = 1, 2, \ldots, r$.

Theorem 3: *Any local optimum of* MAX–rSAT *is such that:*

$$z \geq \max_{k=1,2,\ldots,r} \left[\frac{1}{k+1}\left(km - \sum_{i=1}^{k-1} (k-1)p_i\right)\right]$$

and the bound is sharp.

Proof. Let IUNSAT_i and $\text{IUNSAT}_{\geq i}$ denote the set of unsatisfied clauses with i literals per clause and with at least i literals per clause respectively. Let SAT^1 denote the set of clauses for which a single literal is true: this set can be partitioned into r sets ISAT_i^1 of clauses with i literals for $i = 1, 2, \ldots, r$. Let $\text{ISAT}_{\geq i}^1 = \bigcup_{j=i,i+1,\ldots,r} \text{ISAT}_j^1$. The local optimality conditions $|\text{UNSAT}_j| \leq |\text{SAT}_j|$, $j = 1, 2, \ldots, n$ imply:

$$\sum_j |\text{UNSAT}_j| \leq \sum_j |\text{SAT}_j| = |\text{SAT}^1|.$$

The left-hand-side can be bounded from below by:

$$\sum_{i=1}^{k-1} i|\text{IUNSAT}_i| + k|\text{IUNSAT}_{\geq k}| \qquad \text{for any } k = 1, 2, \ldots, r.$$

As $|\text{UNSAT}| = m - z = \sum_{i=1}^{k-1} |\text{IUNSAT}_i| + |\text{IUNSAT}_{\geq k}|$ we obtain, substituting for $|\text{IUNSAT}_{\geq k}|$ and partitioning SAT^1,

$$k|\text{UNSAT}| \leq \sum_{i=1}^{k-1} (|\text{ISAT}_i^1| + (k-i)|\text{IUNSAT}_i|) + |\text{ISAT}_{\geq k}^1|.$$

Then, as $k|\text{UNSAT}| = k(m-z)$ and $|\text{ISAT}_i^1| + (k-i)|\text{IUNSAT}_i| \leq (k-i)p_i$ for $k > i$ and $|\text{ISAT}_{\geq k}^1| \leq z$, we deduce:

$$k(m-z) \leq \sum_{i=1}^{k-1} (k-i)p_i + z.$$

Hence,

$$z \geq \frac{1}{k+1}\left(km - \sum_{i=1}^{k-1} (k-i)p_i\right). \tag{1}$$

As z is integer and the above formula must hold for all $k = 1, 2, \ldots, r$, the result follows.

To show that the bound is sharp, consider the example:
$C = \{u_1 \vee u_2 \vee \cdots \vee u_r, u_1 \vee \bar{u}_2, u_2 \vee \bar{u}_3, u_3 \vee \bar{u}_1, \bar{u}_1, \bar{u}_2, \bar{u}_3\}$, where $r > 3$. Thus $m = 7$, $p_1 = p_2 = 3$, $p_i = 0$ for $i = 3, 4, \ldots, r-1$ and $p_r = 1$. The solution $U =$ (T, T, T, T, F, F, ..., F) is locally optimal with $z = 4$. The right-hand side of (1) is equal to $\left\lceil \frac{7}{2} \right\rceil = 4$ for $k = 1$, to $\left\lceil \frac{11}{3} \right\rceil = 4$ for $k = 2$, $\left\lceil \frac{k+9}{k+1} \right\rceil \leq 3$ for $k \geq 3$. ■

We now compare the bound just obtained with those of other heuristics. The bound for Johnson's second heuristic, i.e., $z \geq m\frac{2^q - 1}{2^q}$ depends on the minimum number q of literals per clause and is equal to $\frac{m}{2}$ in the general case, i.e., when there are some unit clauses. This is very often the case in logical problems arising in expert systems and data-bases. The bound of Theorem 3 can be rewritten $z \geq \max\left\{\frac{m}{1}, \frac{2m - p_1}{3}, \frac{3m - 2p_1 - p_2}{4}, \ldots\right\}$ and is thus at least as tight as Johnson's bound when $q = 1$. For formulas with exactly q literals per clause, the bound reduces to that of Theorem 1 and is then as weak as that of Johnson's first heuristic.

The bound on the Poljak-Turzik heuristic, i.e., $z \geq \frac{3m - p_1}{4}$, is always greater than the two first terms in the above bound, which are the only ones to be considered for MAX–2SAT. There are thus some MAX-2SAT problems for which the Poljak-Turzik heuristic will always give a solution better than some locally optimum one, regardless of the way the variables are selected.

5. Computational Experience

All the algorithms of the previous sections (except that of Lieberherr as the simpler method of Lieberherr and Specker for MAX–2SAT was already very time consuming) have been implemented in Fortran 77 on a Pyramid 90x and compared on random formulas, characterized by n, m and r where n is the number of variables, m the number of clauses and r the number of literals in a clause. We assume that no clause contains both a literal and its complement.

We have also programmed an exact branch-and-bound algorithm for MAX–2SAT, described in the initial version of this paper [27], to see how close the various heuristics get from the true optimum.

5.1. Analysis of the Performance of Ascent Descent Heuristics

Before comparing the methods presented in sections 3 and 4, we study the effect of the parameter values for both the Simulated Annealing (SA) heuristic and the Steepest Ascent Mildest Descent (SAMD) heuristic.

We first consider in this sensitivity analysis the effect of a (the attenuating coefficient) and t (the mobility parameter) on the SA heuristic. Turning to the SAMD heuristic, we then study the effect of *rep* (maximum number of trials without improvement of the value of the objective function) and of p (number of iterations during which it is forbidden to take a reverse direction) while a and t are at fixed values, i.e., those giving the best results for the SA heuristic.

We use as test problems 10 instances of a MAX–3SAT problem P with $n = 100$ variables and $m = 500$ clauses. For a given set of values of the parameters, we generate $nsol = 50$ random initial points and solve a given instance *nsol* times.

Table 6-1 presents a summary of the results obtained with the SA heuristic for various values of a and t for $rep = 100$ and $rep = 500$. These results are means for 10 problems and $nsol = 50$. Such values are given for the average number of unsatisfied clauses (z), the number of unsatisfied clauses in the best solution among those of the *nsol* runs for each problem (*zopt*), the number of increases of the value of the incumbent (*it*), the number of times the best solution was found (f) and the *cpu* time (*tcpu*), in seconds.

Table 6-1. Sensitivity analysis for SA: influence of a and t.

rep	100					500					
$t \setminus a$	0.6	0.7	0.8	0.9	0.95	0.6	0.7	0.8	0.9	0.95	
	9.5	9.6	8.9	7.5	6.6	5.4	5.7	5.8	4.4	4.1	*z*
	4.7	4.9	4.8	3.7	3.7	3.3	3.3	3.5	3.1	3.2	*zopt*
20	37.4	37.8	38.0	39.6	40.4	42.0	41.5	41.5	42.3	42.5	*iter*
	1.6	1.4	2.2	2.5	3.5	7.2	4.9	4.0	10.8	20.6	*f*
	2.7	3.3	4.7	8.9	17.2	12.2	16.0	23.7	47.1	93.0	*tcpu*
	9.3	9.2	8.7	7.7	6.9	5.4	5.2	5.7	5.2	4.4	*z*
	4.0	4.6	4.7	4.3	3.7	2.8	2.8	3.4	3.5	3.1	*zopt*
50	37.7	38.2	38.3	39.7	40.3	42.0	41.8	42.1	42.2	42.9	*iter*
	2.3	2.1	2.1	3.3	3.0	4.5	4.6	6.6	12.3	12.9	*f*
	3.1	3.8	5.5	10.6	20.7	13.9	18.6	27.6	55.1	110.8	*tcpu*
	9.5	9.6	8.6	7.5	6.3	6.3	5.8	4.6	3.8	4.0	*z*
	4.7	4.8	4.5	4.0	3.1	3.5	3.6	2.7	2.7	3.0	*zopt*
100	37.6	37.8	38.5	39.9	40.3	41.0	41.8	42.7	43.4	42.8	*iter*
	1.8	1.6	2.5	2.1	2.7	3.5	6.0	7.3	18.1	15.2	*f*
	3.3	4.2	6.2	12.2	23.4	15.3	20.4	30.7	61.3	123.0	*tcpu*
	10.0	9.1	8.7	7.9	6.2	5.5	5.4	4.8	4.0	3.8	*z*
	4.7	4.5	4.3	4.0	3.1	2.9	3.1	3.1	2.8	2.6	*zopt*
200	37.6	37.9	39.2	39.5	40.9	41.7	41.9	42.1	42.7	43.0	*iter*
	1.7	1.9	1.7	2.4	2.4	4.6	4.7	9.3	12.7	13.1	*f*
	3.6	4.6	6.7	13.1	25.8	16.5	22.2	33.6	67.5	135.3	*tcpu*

It appears that: (i) the number of unsatisfied clauses decreases significantly when *rep* goes from 100 to 500 and slightly with increasing values of a; it is not significantly affected by t, (ii) computation times are roughly proportional to *rep*, increase with *a*, sharply when *a* approaches 1; they augment slightly with increasing *t*, (iii) the number of iterations increases slightly when *rep* goes from 100 to 500 and does not seem to depend on *t*; (iv) the number *f* of times the best (and presumably optimal in most cases) solution is found does not seem to depend on *t*, increases when *rep* goes from 100 to 500 and sharply increases when *a* tends to 1 and $rep = 500$. This confirm that the SA heuristic often provides an optimal solution, or a solution very close to the optimum, if much computing time is expended.

Table 6-2 presents a summary of the results obtained with the SA and SAMD heuristics for various values of *rep* and *p* for $a = 0.9$ and $t = 100$. It is seen that: i) SAMD gives consistently better results than SA for *p* not too large, in terms of averages (*z*) of the best values found and of times (*f*) such a solution is found, ii) the best value of *p* appears to be around 15, i.e., large enough to allow to force the exploration away from the vicinity of a local maximum, but not so large as to hamper the exploration of regions of high values (taking *p* equal to 15% of the number of variables has also been found to be a good choice in quadratic optimization in 0–1 variables, see Hansen et al. [29]). iii) quality of solutions augments with *rep*, more quickly for SA than for SAMD, iv) *cpu* times of SA are consistently much larger than those of SAMD.

Table 6.2. Sensitivity analysis for SAMD: influence of *rep* and *p*.

	SAMD							SA	
rep \ *p*	10	15	20	25	30	35	40		
	4.9	4.6	4.9	5.3	5.9	6.4	7.1	7.5	*z*
	3.1	3.2	3.1	3.4	3.5	3.6	4.3	4.0	*zopt*
100	33.3	33.4	33.3	32.6	32.0	31.3	31.1	39.9	*iter*
	11.5	15.1	8.0	6.5	3.1	2.6	2.4	2.1	*f*
	0.69	0.73	0.72	0.70	0.69	0.66	0.57	11.91	*tcpu*
	4.6	4.3	4.5	4.9	5.4	6.0	6.5	5.9	*z*
	3.3	3.4	3.4	3.5	3.6	3.7	4.4	3.7	*zopt*
200	34.0	34.4	34.0	33.8	33.0	32.7	32.0	41.3	*iter*
	15.0	20.7	13.0	6.6	3.4	2.5	3.1	6.7	*f*
	1.15	1.20	1.21	1.20	1.21	1.10	1.09	24.45	*tcpu*
	3.4	3.0	3.1	3.6	4.1	4.7	5.3	3.8	*z*
	2.7	2.7	2.7	2.7	2.7	3.1	3.2	2.7	*zopt*
500	34.7	35.2	35.0	34.5	33.9	33.5	32.7	43.4	*iter*
	30.8	39.2	32.4	17.0	7.0	6.5	3.5	18.1	*f*
	2.25	2.33	2.47	2.55	2.51	2.50	2.51	54.34	*tcpu*
	4.0	3.5	3.7	4.0	4.4	4.9	5.4	4.1	*z*
	3.4	3.4	3.4	3.4	3.0	3.1	3.5	3.0	*zopt*
1000	34.0	34.4	34.2	33.7	33.6	33.1	32.6	43.0	*iter*
	31.4	43.5	34.1	22.0	7.7	3.0	3.6	17.1	*f*
	4.04	4.36	4.54	4.63	5.42	5.43	5.00	125.20	*tcpu*
	3.7	3.5	3.5	3.8	4.2	4.5	5.0	3.3	*z*
	3.0	3.4	3.4	3.4	3.4	3.5	3.6	3.0	*zopt*
2000	35.0	34.5	34.4	34.2	33.7	33.4	32.9	44.0	*iter*
	34.3	45.9	42.6	30.6	18.3	9.6	5.5	35.0	*f*
	7.88	8.40	9.06	9.20	9.51	9.62	9.78	224.77	*tcpu*

5.2. Comparison of the Ascent Descent Methods with the Previous Heuristics

5.2.1. General Maximum Satisfiability

The results of experiments with MAX−3SAT and MAX−4SAT problems are summarized in Tables 6-3 and 6-4. The algorithms which are compared are Ascent to the first local optimum (obtained as a byproduct of the Steepest Ascent Mildest Descent algorithm using evaluation function f_0 and noted algorithm *loc* with best optimal value found *zloc*), the heuristics of Johnson, Simulated Annealing and the Steepest Ascent Mildest Descent itself. For each problem size, 50 problems are randomly generated, all clauses being assumed to be equiprobable and possible repetitions being allowed. Each problem is solved 100 times (except for the Johnson's heuristics which are deterministic). The mean values of the best, worst, median and mean values are noted as well as the mean *cpu* time in seconds. The mean number *iter* of improvements of the incumbent is also noted for the Ascent Descent heuristics.

It is seen that: i) the Johnson's heuristics performed poorly, often giving a worst solution that the Ascent heuristics, ii) Steepest Ascent Mildest Descent gives significantly better results than all other heuristics, especially for large problems. Simulated Annealing is the second best heuristic but takes much more time than the

Table 6-3. Comparative results for MAX−3SAT.

n		100	100	100	300	300	300	300	500
m		200	500	700	600	800	1500	2000	5000
zloc	best	0.4	10.8	21.0	2.8	8.3	35.4	64.4	233.8
	worst	4.1	20.6	32.5	9.4	17.2	52.0	85.0	260.0
	median	1.7	15.5	26.0	5.4	11.5	43.2	72.6	251.5
	mean	1.9	15.6	26.6	5.9	12.2	43.5	73.5	249.1
	cpu	0.06	0.12	0.21	0.35	0.42	0.72	0.88	2.33
*zjohn*1	mean	2.3	14.9	28.2	4.7	10.6	45.3	74.1	268.8
	cpu	0.06	0.18	0.33	0.41	0.53	0.69	1.37	3.49
*zjohn*2	mean	1.4	13.5	26.9	2.7	9.0	44.1	76.5	257.4
	cpu	0.06	0.15	0.29	0.35	0.51	0.68	1.26	3.78
zsa	best	0	5.6	15.1	0.7	3.1	22.6	47.6	215.7
	worst	1.1	11.1	21.4	5.5	10.7	41.0	68.1	245.5
	median	0	8.1	17.6	2.3	5.4	28.6	57.4	222.7
	mean	0.2	8.2	18.1	2.7	6.1	30.0	58.0	226.4
	iter	20.1	38.7	44.3	61.2	76.1	109.2	122.4	222.8
	cpu	3.25	4.59	10.58	5.51	6.43	5.62	10.9	15.8
zsamd	best	0	3.7	13.4	0.5	1.2	10.6	34.0	174.6
	worst	1.1	7.4	17.0	4.8	7.6	21.4	45.1	190.6
	median	0.1	4.7	14.4	2.1	4.1	14.7	38.3	182.8
	mean	0.3	5.1	14.7	2.4	4.3	15.3	39	182.8
	iter	16.3	34.0	38.4	48.7	64.0	97.9	113.5	208.7
	cpu	0.48	0.87	1.01	1.02	1.42	3.96	3.39	8.15
	p	15	15	10	25	25	25	25	25

Table 6-4. Comparative results for MAX–4SAT.

n		100	300	300	300
m		700	1500	2500	3000
zloc	best	3.7	4.9	21.2	32.4
	worst	10.9	13.7	37.2	50.7
	median	6.6	7.6	27.5	40.7
	mean	7.1	8.5	28.6	41.6
	cpu	0.20	0.53	0.85	0.94
*zjohn*1	mean	7.0	8.5	29.7	43.9
	cpu	0.34	0.98	1.13	1.42
*zjohn*2	mean	5.2	5.6	27.7	41.2
	cpu	0.32	0.92	1.17	1.36
zsa	best	0.3	1.0	11.9	14.3
	worst	4.0	10.0	25.2	27.3
	median	1.5	3.5	17.2	18.7
	mean	1.8	4.2	17.7	20.0
	iter	31.4	71.2	96.1	114.6
	cpu	11.7	9.94	9.71	21.35
zsamd	best	0.0	0.0	0.4	5.9
	worst	0.7	4.0	7.0	14.6
	median	0.0	0.9	3.4	8.7
	mean	0.1	1.3	3.8	9.4
	iter	24.3	53.1	82.9	99.4
	cpu	0.87	1.69	4.22	7.34
	p	10	25	25	25
	rep	100	100	100	200

other ones, iii) the range of best values found in the 100 runs of the Ascent-Descent heuristics are not very large but however significant. This suggest it may be worthwhile to take the best result of many runs if getting very close to the true optimum is highly desirable.

5.2.2 Maximum 2–Satisfiability

Results for MAX-2SAT problem are summarized in Table 6-5. In addition to the heuristics applied in the general case, those of Lieberherr and Specker, and of Poljak and Turzik as well as the exact algorithm are used. In this series of experiments, problem instances have been kept small in order to allow exact resolution (only two instances of problem with $m = 200$ and $n = 100$ have been solved). Each column corresponds to one problem instance, solved once by the Johnson, Lieberherr-Specker, and Poljak-Turzik heuristics and 100 times by the Ascent Descent heuristics. The algorithms of Lieberherr and Specker, and of Poljak and Turzik, which have the best worst case bounds, give significantly worse results than the other heuristics. The optimum solution obtained after much effort by the exact algorithm is always found at least once by the Ascent Descent heuristics in the 100 resolutions. The average performances of the latter do not differ much.

Table 6-5. Comparative results for MAX–2SAT.

n		10	20	20	50	60	100	100
m		100	200	200	200	200	200	200
zloc	best	13	0	21	13	13	7	8
	worst	13	33	28	26	18	18	19
	mean	13	31	23.2	18.5	15.9	11.3	13.9
	cpu	0.02	0.033	0.03	0.033	0.05	0.08	0.05
*zjohn*1	mean	15	33	24	18	15	12	15
	cpu	0.03	0.05	0.03	0.083	0.07	0.17	0.15
*zjohn*2	mean	15	31	27	18	16	9	12
	cpu	0.03	0.03	0.05	0.067	0.03	0.08	0.07
zlieb	mean	14	32	27	28	20	24	24
	cpu	2.07	369.8	96.5	485.7	1326	4685	4445
zpol	mean	24	45	42	46	40	48	39
	cpu	0.03	0.05	0.05	0.05	0.05	0.08	0.05
zsa	best	13	29	21	13	11	5	7
	worst	13	29	21	16	13	12	12
	mean	13	29	21	13.3	12.2	7.3	8.9
	cpu	0.7	7.47	5.94	4.20	4.19	3.61	3.66
zsamd	best	13	29	21	13	11	5	7
	worst	13	31	21	14	13	15	18
	mean	13	29.8	21	13.1	11.8	8.8	11.5
	cpu	0.03	0.26	0.22	0.19	0.21	0.26	0.24
*z**	mean	13	29	21	13	11	5	7
	p	1	1	4	7	7	7	7
	rep	10	100	100	100	100	100	100

6. Conclusions

The Maximum Satisfiability problem is a natural extension of the basic Satisfiability problem. Several algorithms have been proposed to solve it, some of which have sharp bounds on the guaranteed number of clauses which will be satisfied. We have summarized these heuristics and provided two new ones based on recently proposed Ascent Descent methods (Simulated Annealing and Steepest Ascent Mildest Descent). A systematic experimental study of the performances of all these algorithms shows that the best one appears to be the Steepest Ascent Mildest Descent heuristic. It provided solutions with almost the maximum number of satisfied clauses in all cases where this could be verified, i.e., for the MAX–2SAT problem. Moreover the best solution obtained by that heuristic in a series of 100 resolutions of each of these problems was always optimal, and the computation time remained moderate. For MAX–3SAT and MAX–4SAT, it outperformed all previous heuristics as well as the specialization of the Simulated Annealing algorithmic scheme.

References

[1] Anily and Federgruen, Simulated Annealing methods with general acceptance probabilities, Journal of Applied Probability *24* (1987) 657–667.

[2] Arvind, V., and S. Biswas, An $O(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes all Horn sentences, Information Processing Letters *24* (1987) 67–69.

[3] Asirelli, P., M. de Santis, and A. Martelli, Integrity constraints in logic databases, Journal of Logic Programming *3* (1985) 221–232.

[4] Aspvall, B., M. F. Plass, and R. E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Information Processing Letters *8* (1979) 121–123.

[5] Blair, C. E., R. G. Jeroslow, and J. K. Lowe, Some results and experiments in programming for propositional logic, Computers and Operations Research *13*(5) (1986) 633–645.

[6] Burkard, R. E., and F. Rendl, Thermodynamically motivated simulation procedure, European Journal of Operational Research *17* (1984) 169–174.

[7] Cerny, V., A thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, Journal of Optimization Theory and Applications *45*(1) (1985) 41–51.

[8] Cook, S., The complexity of theorem proving procedures, Proceedings of the Third Symposium of the Association of Computing Machinery on the Theory of Computing (1971) 151–158.

[9] Davis, M., and H. Putnam, A computing procedure for quantification theory, Journal of ACM 7 (1960) 202–215.

[10] Dowling, W. F., and J. H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, Journal of Logic Programming *3* (1984) 267–284.

[11] Dubois, O., Étude de classes de données du problème de satisfiabilité, réfutation de la conjecture de Tovey, Comptes Rendus de l'Académie des Sciences de Paris 303, série I (15) (1986) 765–768.

[12] Edwards, C. S., Some extremal properties of bipartite subgraphs, Canadian Journal of Mathematics *25* (1973) 475–485.

[13] Edwards, C. S., An improved lower bound for the number of edges in a largest bipartite subgraph, in: Recent advances in graph theory (Academia, Prague, 1975), 167–181.

[14] El Baamrani, A., Analyse de deux méthodes heuristiques: méthode de "recuit" et méthode de "plus grande descente—plus petite remontée", B. Sc. Essay, Faculté Universitaire Catholique de Mons, Belgium (1985).

[15] Even, S., Itai, A., and Shamir, A., On the complexity of timetable and multicommodity flow problems, SIAM Journal of Computing *5* (1976) 691–703.

[16] Gallaire, H., Minker, J., and J. M. Nicolas, Logic and databases: A deductive approach, Computing Surveys 16 (2) (June 1984) 153–185.

[17] Garey, R. M., and D. S. Johnson, Computers and intractability, a guide to the theory of NP-completeness, W. H. Freeman and Co. 1979.

[18] Garey, R. M., Johnson, D. S., and L. Stockmeyer, Some simplified NP-complete graph problems, Theoretical Computer Science *1* (1976) 237–267.

[19] Glover, F., Future paths for integer programming and links to artificial intelligence, Computers and Operations Research *13* (5) (1986) 533–549.

[20] Glover, F., Tabu Search—Part 1, ORSA Journal on Computing *1* (1989) 190–206.

[21] Hammer, P. L., Hansen, P., and B. Simeone, Roof duality, complementation and persistency in quadratic 0–1 optimization, Mathematical Programming *28* (1984) 121–155.

[22] Hammer, P. L., and S. Rudeanu, Boolean methods in Operations Research and Related Areas, Springer-Verlag, Heidelberg, 1968.

[23] Hansen, P., Les procédures d'optimisation et d'exploration par séparation et évaluation, in Combinatorial Programming, B. Roy (ed.), Reidel, Dordrecht, 1975, 19–65.

[24] Hansen, P., A cascade algorithm for the logical closure of a set of binary relations, Information Processing Letters 5 (2) (1976) 50–53.

[25] Hansen, P., Methods of nonlinear 0–1 programming, Annals of Discrete Mathematics *5* (1979) 53–69.

[26] Hansen, P., The Steepest Ascent Mildest Descent heuristic for combinatorial programming, presented at the congress on Numerical Methods in Combinatorial Optimization, Capri, March 1986.

[27] Hansen, P., and B. Jaumard, Algorithms for the Maximum Satisfiability Problem, Rutcor Research Report, RRR#43–87, November 1987.

[28] Hansen, P., Jaumard, B., and M. Minoux, A linear expected time algorithm for deriving all logical conclusions implied by a set of boolean inequalities, Mathematical Programming *34* (1986) 223–231.

[29] Hansen, P., S. H. Lu, B. Jaumard and D. Stephany, A computational comparison of the Simulated Annealing and Steepest Descent Mildest Ascent heuristics, (in preparation).
[30] Hayes, F., D. A. Waterman and D. B. Lenat, Building expert systems, Reading, Ma., Addisson-Wesley, 1983.
[31] Henschen, L., and L. Wos, Unit refutation and Horn sets, Journal of ACM *21* (1974) 590–605.
[32] Hertz, A., and D. de Werra, The Tabu Search Metaheuristic: How we used it, Annals of Mathematics and Artificial Intelligence (to appear).
[33] Hooker, J. N., Input proofs and rank one cutting-planes, ORSA Journal on Computing *1* (1989) 137–145.
[34] Jaumard, B., Extraction et utilisation de relations booléennes pour la résolution des programmes linéaires en variables 0–1, Thèse de Doctorat, (Paris, 1986).
[35] Jaumard, B., and B. Simeone, On the complexity of the maximum satisfiability problem for Horn formulas, Information Processing Letters *26* (1987/88) 1–4.
[36] Johnson, D. S., Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences *9* (1974) 256–278.
[37] Jones, N. D., and W. T. Laaser, Complete problems for deterministic polynomial time, Theoretical Computer Science *3* (1977) 107–117.
[38] Karp, R. M., Reducibility among combinatorial problems, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher (eds.), Plenum Press, New York, 1972, 85–104.
[39] Kirpatrick, S., C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science 220 (4598) (1983) 671–674.
[40] Lieberherr, K. J., Algorithmic extremal problems in combinatorial optimization, Journal of Algorithms *3* (1982) 225–244.
[41] Lieberherr, K. J., and E. Specker, Complexity of partial satisfaction, Journal of ACM *28* (2) (1981) 411–421.
[42] Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. *21* (1953) 1087–1092.
[43] Nguyen, T. A., W. A. Perkins, T. J. Laffey and D. Pecora, Checking an expert systems knowledge base for consistency and completeness, IJCAI 85, Arvind Joshi (ed.), Los Altos, California, 375–378.
[44] Poljak, S., and D. Turzik, A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem, Canadian Journal of Mathematics *XXXIV* (3) (1982) 519–524.
[45] Simeone, B., Quadratic 0–1 programming, boolean functions and graphs, Doctoral Dissertation, University of Waterloo (Waterloo, Ontario, 1979).
[46] Schaefer, T., The complexity of satisfiability problems, Proc. 10th Annual ACM Symposium on Theory of Computing (1978) 216–226.
[47] Stephany, D., Localisation de postes en fonction du traffic interpostes, B. Sc. Essay, Faculté Universitaire Catholique de Mons. Belgium (1986).
[48] Sutter, A., Programmation non linéaire en variables 0–1 et application à des problèmes de placement de tâches dans des systèmes distribués, Thèse de Doctorat, Conservatoire National des Arts et Métiers, Paris, France, Juin 1989.
[49] Tovey, C. A., A simplified NP-complete satisfiability problem, Discrete Applied Mathematics *8* (1984) 85–89.
[50] Yamasaki, S., and S. Doshita, The satisfiability problem for a class of Horn sentences and some non-Horn sentences in propositional logic, Information and Control *59* (1983) 1–12.

P. Hansen
Rutcor, Hill Center for Mathematical Sciences
Rutgers University
New Brunswick, NY 08903
U.S.A.

Brigitte Jaumard
Gérad and École Polytechnique
Département de Mathématiques Appliquées
Montréal, Québec, C.P. 6079, Succ. A.
Canada, H3C 3A7