Annals of Operations Research, 25 (1990) 43–58

COMPUTATIONAL EXPERIENCE WITH AN INTERIOR POINT ALGORITHM ON THE SATISFIABILITY PROBLEM

A.P. KAMATH, N.K. KARMARKAR, K.G. RAMAKRISHNAN and M.G.C. RESENDE

Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA

Abstract

We apply the zero–one integer programming algorithm described in Karmarkar [12] and Karmarkar, Resende and Ramakrishnan [13] to solve randomly generated instances of the satisfiability problem (SAT). The interior point algorithm is briefly reviewed and shown to be easily adapted to solve large instances of SAT. Hundreds of instances of SAT (having from 100 to 1000 variables and 100 to 32,000 clauses) are randomly generated and solved. For comparison, we attempt to solve the problems via linear programming relaxation with MINOS.

Keywords: Integer programming, interior point method, logic, satisfiability.

1. Introduction

We consider here the satisfiability problem (SAT) in propositional calculus, a central problem in mathematical logic that was posed as the original NP-complete problem [3].

A Boolean *variable* x is a variable that can assume only the values *true* or *false*. Boolean variables can be combined by the logical connectives *or* ($\vee$), *and* ($\wedge$) and *not* ($\bar{x}$) to form Boolean formulae. A variable or a single negation of the variable is called a *literal*. A Boolean formula consisting of only literals combined by just the $\vee$ connector is called a *clause*.

The satisfiability problem (SAT) can be defined as follows. Given n clauses $\mathscr{C}_1, \ldots, \mathscr{C}_n$ involving m variables $x_1, \ldots, x_m$, is the formula

$$\mathscr{C}_1 \wedge \mathscr{C}_2 \wedge \ldots \wedge \mathscr{C}_n \tag{1}$$

satisfiable? What is wanted is an assignment of truth values to the Boolean variables so that the Boolean formula (1) has value *true*. Hence, the assignment of truth values to the Boolean variables must make each clause $\mathscr{C}_1, \mathscr{C}_2, \ldots, \mathscr{C}_n$ *true*. We present a method for finding a satisfiable truth assignment, based on the interior point zero–one integer programming algorithm described in Karmarkar [12] and Karmarkar, Resende and Ramakrishnan [13]. Once we have a truth

assignment, it is trivial to prove satisfiability by substitution. In case the formula is not satisfied, our method does not construct a proof of nonsatisfiability.

Consider the following three clauses,

$$\mathscr{C}_1 = x_1 \vee \bar{x}_2 \vee x_3,$$
$$\mathscr{C}_2 = x_2 \vee x_3,$$
$$\mathscr{C}_3 = x_1 \vee \bar{x}_3.$$

The formula $\mathscr{C}_1 \wedge \mathscr{C}_2 \wedge \mathscr{C}_3$ is clearly satisfiable, since the truth assignment $x_1 = \textit{true}$, $x_2 = \textit{true}$ and $x_3 = \textit{false}$ makes each clause $\mathscr{C}_1$, $\mathscr{C}_2$ and $\mathscr{C}_3$ *true*.

There has been great interest in devising efficient algorithms for solving the satisfiability problem. One obvious way would be to try all possible truth assignments to see if one satisfies the formula. This has exponential complexity, since in the worst case one may have to try 2^m truth assignments. The Davis–Putnam procedure [4] is a technique that has been shown to take polynomial average time complexity, subject to certain restrictions. Unfortunately, these constraints substantially limit the application of the algorithm. Resolution is a widely used method for inference problems in propositional calculus [19]. Unfortunately, resolution not only has exponential worst case complexity but fails to solve even moderately sized inference problems. The resolution method can find unsatisfiability quickly in certain classes of problems but is slow when it comes to verifying satisfiability, as all possible resolutions need to be tried out before concluding that the inference relation holds or that the problem is satisfiable. Hooker [9] has shown that mathematical programming methods are substantially faster than resolution.

An interesting aspect of the problem is that it can be formulated as an integer program. The integer programming formulation is immediate if one identifies logical value *true* with the integer 1 and *false* with -1. The following procedure is standard to transform a clause into a linear inequality. First, all $\vee$ connectives are transformed into the $+$ of ordinary addition. The literal x is represented by the integer variable w and $\bar{x}$ is transformed into a $-w$. We impose the condition that each clause is *true*. Hence, if a clause $\mathscr{C}$ has $n(\mathscr{C})$ literals then at least one must be *true*, i.e. contribute to a $+1$, while the rest are free to be *true* or *false*, thus contributing at least -1 each, resulting in the constraint

$$\sum_{j \in I_{\mathscr{C}}} w_j - \sum_{j \in J_{\mathscr{C}}} w_j \geqslant 2 - n(\mathscr{C}),$$

where $I_{\mathscr{C}}$ is the set of indices of all $+w$ variables in clause $\mathscr{C}$, $J_{\mathscr{C}}$ is the set of indices of all $-w$ variables in $\mathscr{C}$ and $n(\mathscr{C}) = |I_{\mathscr{C}}| + |J_{\mathscr{C}}|$. Hence, the clauses in the earlier example would give us the constraints

$$w_1 - w_2 + w_3 \geqslant -1,$$
$$w_2 + w_3 \geqslant 0,$$
$$w_1 - w_3 \geqslant 0,$$

with the restriction that $w_j = \pm 1$, $j = 1, 2, 3$, requiring that extra constraints be added to insure that each w_j be in the closed interval $[-1, 1]$. These constraints are

$$-1 \leqslant w_1 \leqslant 1,$$
$$-1 \leqslant w_2 \leqslant 1,$$
$$-1 \leqslant w_3 \leqslant 1.$$

This ± 1 integer programming formulation of SAT is similar to the more common form of integer programming, where variables x_j take on (0, 1) values. The ± 1 problem can be transformed into the (0, 1) form with the change of variables

$$x_j = \frac{1 + w_j}{2}, \quad j = 1, \ldots, m.$$

Several authors have investigated integer programming approaches to resolving Boolean logic problems, e.g. [1,5–11,21–23]. A recent survey is given in [2].

Recently a cutting plane algorithm that uses resolvents as cuts has been introduced by Hooker [9], who has reported good computational results for certain problem classes. However, the method, based on the simplex method of linear programming, has been described ineffective on dense satisfiability problems. Moreover, as will be discussed later, Hooker allows for the presence of unit clauses in his sample problems which contribute toward making the problems less difficult than instances of SAT in which single literal clauses are disallowed. Hooker reports computational results for small sized problems and the behavior of his method for larger instances is hard to predict.

An outline of the remainder of this paper is as follows. In section 2 we briefly review the integer programming algorithm. Specific details required to apply the integer programming algorithm to SAT are given in section 3. Computational results are provided in section 4. In section 5 we make concluding remarks.

2. Zero–one integer programming

In this section, we briefly review the interior point algorithm used in this study. We omit all proofs. They can be found in [12] and [14]. We consider the following integer programming problem:

INTEGER PROGRAMMING: Let $B \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^n$. Find $w \in \mathbb{R}^m$ such that:

$$B^{\mathrm{T}} w \leqslant b, \tag{2}$$
$$w_i = \pm 1, \quad i = 1, \ldots, m. \tag{3}$$

We propose an interior point approach to solve (2), (3), i.e. a heuristic that generates a sequence of points $\{w^0, w^1, \ldots, w^k, \ldots\}$ where for all $k = 0, 1, \ldots$

$$w^k \in \{ w \in \mathbb{R}^m \mid B^{\mathrm{T}} w < b;\ -e < w < e \},$$

where $e^T = (1, \ldots, 1)$. In practice, this sequence often converges to a point from which one can round off to a ± 1 integer solution to (2)–(3). No guarantee can be made as to whether the heuristic will be successful, but in this paper we provide several large instances of SAT where it succeeds in proving satisfiability.

To simplify notation, let I denote an $m \times m$ identity matrix,

$$A = [B \vdots I \vdots -I]$$

and

$$c = \begin{bmatrix} b \\ 1 \\ \vdots \\ 1 \end{bmatrix},$$

and let

$$\mathscr{I} = \{ w \in \mathbb{R}^m \mid A^T w \leq c \text{ and } w_i = \pm 1 \}.$$

With this notation, INTEGER PROGRAMMING can be restated as: Find $w \in \mathscr{I}$.

Throughout this paper we assume that A^T has full rank. Let

$$\mathscr{L} = \{ w \in \mathbb{R}^m \mid A^T w \leqslant c \}$$

and consider the linear programming relaxation of (2)–(3), i.e. find $w \in \mathscr{L}$. One way of selecting ± 1 integer solutions over fractional solutions in linear programming is to introduce the quadratic objective function,

$$\text{maximize } w^T w = \sum_{i=1}^{m} w_i^2 \tag{4}$$

and solve the NP-complete [20] nonconvex quadratic programming problem

$$\text{subject to: } A^T w \leqslant c. \tag{5}$$

The following proposition establishes the relationship between (4)–(5) and INTEGER PROGRAMMING.

PROPOSITION 2.1

Let $w \in \mathscr{L}$. Then $w \in \mathscr{I} \Leftrightarrow w^T w = m$, where m is the optimal solution to (4)–(5).

We now consider an algorithm to solve (4)–(5) and show how to apply this algorithm to integer programming. Let

$$w^0 \in \mathscr{L}_s = \{ w \in \mathbb{R}^m \mid A^T w < c \}$$

be a given initial interior point. The algorithm generates a sequence of interior points of $\mathscr{L}$. Let $w^k \in \mathscr{L}_s$ be the kth iterate. Around w^k we construct a

quadratic approximation of the potential function

$$\phi(w) = \log\sqrt{m - w^{\mathrm{T}}w} - \frac{1}{n}\sum_{k=1}^{n} \log\, d_k(w),$$

where

$$d_k(w) = c_k - a_k^{\mathrm{T}}w, \quad k = 1, \ldots, n.$$

Let $D = \mathrm{diag}(d_1(w), \ldots, d_n(w))$, $e = (1, \ldots, 1)$, $f_0 = m - w^{\mathrm{T}}w$ and C be a constant. The quadratic approximation of $\phi(w)$ around w^k is given by

$$Q(w) = \tfrac{1}{2}(w - w^k)^{\mathrm{T}}H(w - w^k) + h^{\mathrm{T}}(w - w^k) + C, \tag{6}$$

where the Hessian is

$$H = -\frac{2}{f_0}I - \frac{4}{f_0^2}w^k w^{k^{\mathrm{T}}} + \frac{1}{n}AD^{-2}A^{\mathrm{T}} \tag{7}$$

and the gradient is

$$h = -\frac{1}{f_0}w^k + \frac{1}{n}AD^{-1}e. \tag{8}$$

Minimizing (6) subject to $A^{\mathrm{T}}w \leqslant c$ is NP-complete. However, if the polytope is approximated by an inscribed ellipsoid,

$$\mathscr{E}(r) = \left\{w \in \mathbb{R}^m \mid (w - w^k)^{\mathrm{T}}AD^{-2}A^{\mathrm{T}}(w - w^k) \leqslant r^2 \leqslant 1\right\}, \tag{9}$$

the resulting approximate problem,

$$\text{minimize } \tfrac{1}{2}(\Delta w)^{\mathrm{T}}H\Delta w + h^{\mathrm{T}}\Delta w \tag{10}$$

$$\text{subject to: } (\Delta w)^{\mathrm{T}}AD^{-2}A^{\mathrm{T}}(\Delta w) \leqslant r^2 \leqslant 1, \tag{11}$$

where $\Delta w \equiv w - w^k$, is easy.

We now consider an algorithm to solve INTEGER PROGRAMMING based on nonconvex quadratic programming. The approach used to solve the nonconvex optimization problem

$$\text{minimize}\left\{\phi(w) \mid A^{\mathrm{T}}w \leqslant c\right\},$$

is similar to the classical Levenberg–Marquardt methods [16,17], first suggested in the context of nonlinear least squares. This algorithm solves (10)–(11) to produce a descent direction Δw^* for the potential function $\phi(w)$. A solution $\Delta w^* \in \mathbb{R}^m$ to (10)–(11) is optimal if and only if there exists $\mu \geqslant 0$ such that:

$$\Delta w^*\left(H + \mu AD^{-2}A^{\mathrm{T}}\right) = -h, \tag{12}$$

$$\mu\left((\Delta w^*)^{\mathrm{T}}AD^{-2}A^{\mathrm{T}}\Delta w^* - r^2\right) = 0, \tag{13}$$

$$H + \mu AD^{-2}A^{\mathrm{T}} \text{ is positive semi-definite.} \tag{14}$$

With the change of variables $\gamma = 1/(\mu + 1/n)$ and substituting (7) and (8) into (12) we obtain an expression for Δw^* satisfying (12):

$$\Delta w^* = -\left(AD^{-2}A^{\mathrm{T}} - \frac{4\gamma}{f_0^2} w^k w^{k\mathrm{T}} - \frac{2\gamma}{f_0} I\right)^{-1} \gamma\left(-\frac{1}{f_0} w^k + \frac{1}{n} AD^{-1}e\right). \tag{15}$$

Note that r does not appear in (15). However, (15) is not defined for all values of r. It can be shown that if the radius r of the ellipsoid (11) is kept within a certain size, then there exists an interval $0 \leqslant \gamma \leqslant \gamma_{\max}$ such that

$$AD^{-2}A^{\mathrm{T}} - \frac{4\gamma}{f_0^2} w^k w^{k\mathrm{T}} - \frac{2\gamma}{f_0} I \tag{16}$$

is nonsingular. The following proposition establishes a descent direction of $\phi(w)$.

PROPOSITION 2.4

There exists $\gamma > 0$ such that the direction Δw^*, given in (15), is a descent direction of $\phi(w)$.

We now outline the steps of the algorithm to find a solution of (10)–(11) satisfying conditions (12)–(14) and show how to incorporate this approach into an algorithm to solve INTEGER PROGRAMMING. Each iteration of this algorithm is comprised of two tasks. To simplify notation, let

$$H_c = AD^{-2}A^{\mathrm{T}}, \tag{17}$$

$$H_o = -\frac{4}{f_0^2} w^k w^{k\mathrm{T}} - \frac{2}{f_0} I, \tag{18}$$

and define

$$M = H_c + \gamma H_o.$$

Given the current iterate w^k we first seek a value of γ such that $M\Delta w = \gamma h$ has a solution Δw^*. This can be done by binary search, as we will see shortly. Once such a parameter γ is found, the linear system

$$M\Delta w^* = \gamma h \tag{19}$$

is solved for $\Delta w^* \equiv \Delta w^*(\gamma(r))$. It is easy to verify that the *length*,

$$l(\Delta w^*(\gamma)) = (\Delta w^*(\gamma(r)))^{\mathrm{T}} AD^{-2}A^{\mathrm{T}} \Delta w^*(\gamma(r)),$$

is a monotonically increasing function of γ in the interval $0 \leqslant \gamma \leqslant \gamma_{\max}$.

Optimality condition (13) implies that $r = \sqrt{l(\Delta w^*(\gamma))}$ if $\mu > 0$. Small lengths result in small changes in the potential function, since r is small and the optimal solution lies on the surface of the ellipsoid. A length that is too large may not correspond to an optimal solution of (10)–(11), since this may require $r > 1$. We maintain an interval $(\underline{l}, \bar{l})$ called the *acceptable length region* and accept a length $l(\Delta w^*(\gamma))$ if $\underline{l} \leqslant l(\Delta w^*(\gamma)) \leqslant \bar{l}$. If $l(\Delta w^*(\gamma)) < \underline{l}$, γ is increased and (19) is

Pseudo code 2.1
The **ip** algorithm

```
procedure ip(A, c, γ0, l, l̄0)
1  k := 0; γ := γ0; l := l; l̄ := l̄0, K := 0;
2  w^k := get_start_point(A, c);
3  w̃^k := round_off(w^k);
4  do A^T w̃^k ≱ c →
5     Δw* := descent_direction(γ, w^k, l, l̄);
6     do φ(w^k + αΔw*) ≥ φ(w^k) and l̄ > ε →
7        l̄ := l̄/l̄_r;
8        Δw* := descent_direction(γ, w^k, l, l̄)
9     od;
10    if φ(w^k + αΔw*) < φ(w^k) →
11       w^{k+1} := w^k + αΔw*;
12       w̃^{k+1} := round_off(w^{k+1});
13       k := k + 1
14    fi;
15    if l̄ ≤ ε →
16       print("Converged to local minimum.");
17       exit()
18    fi
19 od
end ip;
```

Pseudo code 2.2
The **descent_direction** algorithm

```
procedure descent_direction(γ, w^k, l, l̄)
1  l := ∞; LD_key := false; γ̄_key := false; γ_key := false;
2  do l > l̄ or (l < l and LD_key = false) →
3     M := H_c + γH_0; b := γh;
4     do MΔw = b has no solution →
5        γ := γ/γ_r; LD_key := true;
6        M := H_c + γH_0; b := γh
7     od;
8     Δw* := M^{-1}b; l := (Δw*)^T A D^{-2} A^T Δw*;
9     if l < l and LD_key = false →
10       γ := γ; γ_key := true;
11       if γ̄_key = true → γ := √(γ γ̄) fi;
12       if γ̄_key = false → γ := γ · γ_r fi
13    fi;
14    if l > l̄ →
15       γ̄ := γ; γ̄_key := true;
16       if γ_key = true → γ := √(γ γ̄) fi;
17       if γ_key = false → γ := γ/γ_r fi
18    fi
19 od;
20 do l < l and LD_key = true → l := l/l_r od;
21 return(Δw*)
end descent_direction;
```

resolved with the new M matrix and h vector. On the other hand, if $l(\Delta w^*(\gamma)) > \bar{l}$, γ is reduced and (19) is resolved. We shall say a *local minimum* has been found if the length is reduced below a tolerance ϵ. If a local minimum is found, several strategies can be considered. In one approach, the problem is modified (by adding a cut, for example) and the algorithm is applied to the new problem. Once an acceptable length is produced the new iterate w^{k+1} is computed by moving in direction $\Delta w^*(\gamma)$ from w^k with a step size $\alpha < 1$,

$$w^{k+1} = w^k + \alpha \Delta w^*(\gamma). \tag{20}$$

The current iterate w^{k+1} is rounded off to the nearest ± 1 vertex: $\tilde{w}^{k+1} = (\pm 1, \ldots, \pm 1)$. If $\tilde{w}^{k+1}$ is such that $A^T \tilde{w}^{k+1} \leqslant c$ then $\tilde{w}^{k+1}$ is a global optimal solution of (4)–(5).

Pseudo code 2.1 details procedure **ip**, the integer programming algorithm that makes use of procedure **descent_direction** to optimize (10)–(11) producing the descent direction. In **ip**, procedure **get_start_point** returns an initial starting interior point and procedure **round_off** rounds a fractional solution to a ± 1 integer solution. These procedures are discussed in more detail in section 3.

Pseudo code 2.2 details procedure **descent_direction**.

3. Application specific details

In this section, we discuss algorithmic details specific to this application (SAT).

3.1. INITIAL SOLUTION

The algorithm requires an initial interior point solution to

$$\sum_{j \in I_{\mathscr{C}_i}} w_j - \sum_{j \in J_{\mathscr{C}_i}} w_j \geqslant 2 - n(\mathscr{C}_i), \quad i = 1, \ldots, n, \tag{21}$$

$$-1 \leqslant w_j \leqslant 1, \quad j = 1, \ldots, m. \tag{22}$$

One way to obtain such a starting solution is to solve a Phase I artificial linear program and use its solution as the initial solution for the integer programming algorithm. Instead, we simply use the origin $w^0 = (0, \ldots, 0)$ as the initial solution because, in practice, there is no apparent benefit from the linear programming approach. However, the origin is not an interior point to (21)–(22) if there exists any clause $\mathscr{C}_i$ such that $n(\mathscr{C}_i) = 2$. To get around this problem, we use an alternative right hand side to (21), resulting in

$$\sum_{j \in I_{\mathscr{C}_i}} w_j - \sum_{j \in J_{\mathscr{C}_i}} w_j \geqslant 1 - n(\mathscr{C}_i), \quad i = 1, \ldots, n, \tag{23}$$

$$-1 \leqslant w_j \leqslant 1, \quad j = 1, \ldots, m. \tag{24}$$

PROPOSITION 3.1

If $n(\mathscr{C}_i) \geq 2$, $i = 1, \ldots, n$ and $w_j = \pm 1$, $j = 1, \ldots, m$, then (23)–(24) is a valid description of SAT.

Proof

We first show that $w^0 = (0, \ldots, 0)$ satisfies the constraints. Constraints (24) are clearly satisfied. Constraints (23) are also satisfied, because

$$\sum_{j \in I_{\mathscr{C}_i}} w_j^0 - \sum_{j \in J_{\mathscr{C}_i}} w_j^0 = 0 > 1 - n(\mathscr{C}_i), \quad i = 1, \ldots, n,$$

since $n(\mathscr{C}_i) \geq 2$, $i = 1, \ldots, n$. We must now show that for all ± 1 integer solutions, at least one $w_j = 1$ for $j \in I_{\mathscr{C}_i}$ or at least one $w_j = -1$ for $j \in J_{\mathscr{C}_i}$ for $i = 1, \ldots, n$. Assume the contrary, i.e. assume $\forall j \in I_{\mathscr{C}_i}$, $w_j = -1$ and $\forall j \in J_{\mathscr{C}_i}$, $w_j = 1$. Then $-|I_{\mathscr{C}_i}| - |J_{\mathscr{C}_i}| \geq 1 - |I_{\mathscr{C}_i}| - |J_{\mathscr{C}_i}|$, which implies $0 \geq 1$, a contradiction. □

3.2. ROUNDING OFF

Rounding off can be done in a straightforward manner, for $j = 1, \ldots, m$,

$$\tilde{w}_j = \begin{cases} +1 & \text{if } w_j > 0, \\ -1 & \text{if } w_j \leq 0. \end{cases}$$

We call this rounding scheme *type A*.

Rounding scheme A does not discriminate between, say, positive interior point variables $w_{k_0} = 0.01$ and $w_{k_1} = 0.99$, assigning both variables a $+1$. Intuitively however, one should expect the Boolean variable x_{k_1} to be *true* with higher probability than variable x_{k_0}. Furthermore, scheme A takes no advantage of the structure of the constraints. Assigning a truth value to a Boolean variable may force some other variable to take on a certain truth value. For example, if clause $\mathscr{C}$: $\bar{x}_1 \vee \bar{x}_2$ is one of the clauses to be satisfied and variable x_1 is set to *true*, then variable x_2 will necessarily have to be assigned a *false* value.

Considering the above observations, we propose a second rounding procedure, called scheme *type B*. This scheme has two possible outcomes. Either it produces a satisfiable truth assignment and hence indicates that the interior point algorithm should halt, or it does not find a satisfiable truth assignment, indicating that the interior point algorithm should proceed. Rounding scheme B can be described by the following five step procedure.

STEP 0: Sort all interior point variables w_j, $j = 1, \ldots, m$, in decreasing order of their absolute values and place the sorted variables in a priority queue

$$\mathscr{Q} = \{w_{j_1}, w_{j_2}, \ldots, w_{j_m}\},$$

where

$$|w_{j_1}| \geq |w_{j_2}| \geq \cdots \geq |w_{j_m}|.$$

STEP 1: Select the first variable from $\mathcal{Q}$, say w_k and set

$$\tilde{w}_k = \begin{cases} +1 & \text{if } w_k > 0, \\ -1 & \text{if } w_k \leqslant 0. \end{cases}$$

Remove variable w_k from $\mathcal{Q}$.

STEP 2: The Boolean variable x_k, corresponding to w_k, is assigned the appropriate value, i.e.

$$x_k = \begin{cases} \textit{true} & \text{if } \tilde{w}_k = +1, \\ \textit{false} & \text{if } \tilde{w}_k = -1. \end{cases}$$

STEP 3: Consider all clauses where x_k occurs, negated or otherwise. As a consequence of the assignment of the truth value to x_k some of these clauses may become satisfied, while others may be reduced. If there is any single literal clause in the reduced set of clauses, that Boolean variable is forced to take on a definite value. For example, if $\mathcal{C}$: $\bar{x}_1$ is a single literal clause, x_1 is forced to be *false*. Remove that variable from $\mathcal{Q}$. Set the Boolean variable to its required value. As a consequence, some clauses may be made satisfiable, some may be reduced. Furthermore, there may exist two clauses for which a variable must take on contradicting values. For example, if $x_1 = \textit{false}$ in $\mathcal{C}_1$: $x_1 \vee x_2$ and $\mathcal{C}_2$: $x_1 \vee \bar{x}_2$, then any value given to x_2 will cause either $\mathcal{C}_1$ or $\mathcal{C}_2$ to become unsatisfiable. If a contradiction is found, terminate the rounding procedure, indicating that a satisfiable truth assignment has not been found and proceed with the interior point algorithm. Note that the current state w^k of the interior point algorithm is not affected if the rounding scheme fails to produce a satisfiable truth assignment. STEP 3 is recursively executed until there are no more single literal clauses to process.

STEP 4: If all clauses are satisfied, terminate the rounding procedure and the integer programming algorithm with a satisfiable truth assignment. Else, go to STEP 1.

3.3. LOCAL MINIMA

The integer programming algorithm is not guaranteed to converge to a satisfiable solution. When the algorithm converges to a local minimum that is not global one could do as in [14] or [15]. We have not yet implemented any such scheme for this class of problems and therefore simply stop and say the algorithm failed to find a feasible truth assignment.

4. Computational results

We now describe the computational experiments. We have generated hundreds of random instances of SAT using a model similar to the one described by Hooker [9,10].

As input to the random instance generator we provide the number of clauses n, the number of variables m and the expected number of literals per clause k. Elements a_{ij} of the constraint matrix A are assigned value $+1$ with probability $p/2$, -1 with probability $p/2$ and 0 with probability $1-p$, where p is such that the expected number of nonzero elements per clause is k. All null clauses are rejected. We generate no single literal clauses, differing here from [9], since those clauses can be trivially removed. By removing single literal clauses, simple linear programming relaxation can no longer be used to prove unsatisfiability. In [10], Hooker also discards single literal clauses.

All runs were carried out on *mhkorbx*, a KORBX® * parallel/vector computer. The KORBX® uses a variant of the UNIX® Operating System, called the KORBX® Operating System. The KORBX® machine we used operates in scalar mode at approximately 1 MFlops and at 32 MFlops with full vector concurrent mode. In our experiments all code was written in FORTRAN and C and compiled on the KORBX® *fortran* compiler with optimization flags $-$O $-$DAS and KORBX® *CC* compiler with optimization flag $-$O. No special care was taken to vectorize or parallelize the code. All times reported are user CPU times given by the system call *times*().

We report for each problem class the number of instances proven satisfiable, the number of instances in which the algorithm converged to a local minimum, the minimum, average and maximum number of iterations and the minimum, average and maximum solution time of the integer programming algorithm.

We used the following algorithm parameter settings for all problem instances: $\gamma_0 = 10$, $l_0 = 0.5$, $\bar{l}_0 = 1.0$, $\epsilon = 10^{-12}$, $\bar{l}_r = 4$, $\alpha = 0.5$, $\gamma_r = \sqrt{2}$ and $l_r = 4$. Our implementation uses a preconditioned conjugate gradient algorithm to solve (19) at each iteration. The conjugate gradient algorithm stops when $|1 - \cos\theta| < 10^{-8}$, where θ is the angle between $M\Delta w$ and h.

Table 1 summarizes the computational results for the integer programming algorithm using rounding scheme A. There, results are tabulated for instances varying from 50 variables by 100 clauses, with an average of 5 literals per clause to 1000 variables by 2000 clauses and an expected number of 15 literals per clause. Statistics for iterations and solution times include only instances in which the algorithm converged to a global minimum. Times are in seconds.

* KORBX and UNIX are registered trademarks of AT&T.

Table 1
Computational results (rounding scheme A)

Problem			Instances		Iterations			Solution time		
vars	clauses	$E[n(\mathscr{C})]$	global min	local min	min	mean	max	min	mean	max
50	100	5	89	11	1	46.2	530	0.5	9.4	67.0
100	200	5	82	18	1	136.2	1002	1.0	50.9	297.3
200	400	7	81	19	1	412.6	2070	2.3	313.7	1450.7
400	800	10	86	14	1	44.2	1320	4.8	63.6	1577.9
500	1000	10	80	20	1	119.4	2350	6.0	230.9	4464.5
1000	2000	10	6	4	1	1450.0	3374	18.2	4021.2	8183.2
1000	2000	11	8	2	1	174.6	659	13.6	715.5	2032.1
1000	2000	12	9	1	1	194.9	1535	13.6	571.0	4052.0
1000	2000	13	10	0	1	78.0	7710	14.7	215.4	2008.0
1000	2000	15	9	1	1	1.0	1	15.0	16.5	19.8

We can make the following observations about the runs in table 1:
- The interior point method produced satisfiable truth assignments for the majority of instances.
- All problem classes (rows in the table) had at least one instance for which the algorithm produced a satisfiable solution in one iteration.
- For the problems with $m = 1000$ variables and $n = 2000$ clauses, the interior point algorithm produces satisfiable truth assignments in more number of cases as the expected number of literals per clause increases, i.e. as the problem becomes easier.

We reran all instances in which the integer programming algorithm with rounding scheme A took more than one iteration, this time using rounding scheme B. We also ran larger and comparatively more difficult problems with this scheme. These had 1000 variables and had from 2000 to 32,000 clauses. Those results are shown in table 2.

We can make the following observations about the runs in table 2:
- Scheme B went to a non-integer solution in very few (only 2) cases.
- The majority of instances were resolved in only one iteration.
- All ten instances of size $m = 1000$ variables and $n = 32{,}000$ clauses were resolved in less than 6 CPU minutes.
- All problem classes (rows in the table), with the exception of ($m = 1000$, $n = 4000$, $E[n(\mathscr{C})] = 4$) had instances that were solved in a single iteration. Only two instances were tested for the class that had no single iteration solution.

In [9], Hooker reports that by solving a linear programming relaxation of SAT one frequently produces an integer solution. We have used MINOS 5.1 [18] to

Table 2
Computational results (rounding scheme B)

Problem			Instances		Iterations			Solution time		
vars	clauses	$E[n(\mathscr{C})]$	global min	local min	min	mean	max	min	mean	max
50	100	5	52	0	1	1.7	35	0.5	0.7	7.7
100	200	5	70	0	1	1.0	1	1.0	1.1	1.7
200	400	7	69	0	1	1.0	1	2.4	3.5	6.3
400	800	10	31	0	1	1.0	1	5.1	5.6	7.7
400	800	7	20	0	1	1.0	1	4.7	7.8	16.1
500	1000	10	49	0	1	1.0	1	6.5	7.4	9.8
1000	2000	10	10	0	1	1.0	1	14.9	18.5	20.5
1000	2000	7	50	0	1	1.0	1	18.1	21.5	27.3
1000	2000	3	49	1	1	5.8	159	8.7	50.4	1537.8
1000	4000	10	10	0	1	1.0	1	24.0	25.1	25.8
1000	4000	4	1	1	110	110.0	110	1085.4	1085.4	1085.4
1000	8000	10	10	0	1	1.0	1	37.5	38.0	38.8
1000	16000	10	10	0	1	1.5	6	46.2	66.4	92.1
1000	32000	10	10	0	1	24.4	235	80.1	232.4	311.3

solve the linear programming relaxation

minimize z_0

subject to:

$$z_0 + \sum_{j \in I_{\mathscr{C}_i}} z_j - \sum_{j \in J_{\mathscr{C}_i}} z_j \geqslant 1 - |J_{\mathscr{C}_i}|, \quad i = 1, \ldots, n,$$

$$0 \leqslant (z_0, z_j) \leqslant 1, \quad j = 1, \ldots, m,$$

corresponding to some of the smaller problems tested on the integer programming algorithm. Table 3 summarizes the results.

We can make the following observations about the runs in table 3:

- Contrary to Hooker [9], where single literal clauses were admitted, the simplex method failed to find integral solutions to the linear programming relaxations

Table 3
Computational results – linear programming relaxation

Problem			Instances		Iterations			Solution time		
vars	clauses	$E[n(\mathscr{C})]$	int	frac	min	mean	max	min	mean	max
50	100	5	23	77	46	93.4	139	1.6	3.5	5.5
50	100	10	93	7	61	131.5	219	2.6	5.8	10.0
100	200	5	11	89	157	355.1	574	10.7	24.7	41.0
200	400	7	23	77	1011	1523.5	2042	142.9	231.1	316.2
400	800	10	2	3	4799	5610.0	6593	1285.9	1790.0	2147.0
400	800	7	0	15	4982	6143.3	7486	1616.3	2050.4	2802.7

Table 4
Computational results – CPU time ratios

Problem			CPU time ratios
vars	clauses	$E[n(\mathscr{C})]$	
50	100	5	5.0
100	200	5	22.5
200	400	7	66.0
400	800	7	262.9
400	800	10	319.6

in the majority of instances tested. In fact, for $n = 400$ variables, $m = 800$ clauses and $E[n(\mathscr{C})] = 7$ literals per clause, the simplex method failed in all 15 instances. To achieve the performance reported by Hooker for ($n = 50$, $m = 100$, $E[n(\mathscr{C})] = 5$), we had to double $E[n(\mathscr{C})]$ to 10.

- Large instances could not be solved because of excessive solution times.
- The average solution time ratios between MINOS and the interior point method (using rounding scheme B) are shown in table 4. Because of these time ratios, we feel that simplex based branch and cut methods will encounter much difficulty to outperform this interior point approach, since at least one linear programming relaxation must be solved.

5. Concluding remarks

An interior point algorithm for integer programming [12,13] has been applied to find truth assignments in instances of SAT. The algorithm finds truth assignments in the majority of large instances (up to 1000 variables and 32,000 clauses) with the straightforward rounding scheme A and takes very few iterations with the more involved rounding scheme B. We cannot, however, guarantee that the algorithm will find a truth assignment if one exists. Moreover, we cannot make any conclusion when the algorithm converges to a local minimum without finding a satisfiable truth assignment.

Large instances of SAT were solved in little CPU time. In general, MINOS 5.1 required more CPU time to solve only a linear programming relaxation of the problem, in which fractional solutions were frequently found.

In this implementation of the interior point algorithm, no special treatment was given to local minima. We have observed in the computational experiments described in this paper, that when a local minimum is encountered, it is usually the case that very few constraints remain unsatisfied. Future work will focus on treating local minima. One possible approach is to add cuts and restart the algorithm, as in [14]. Another is to use a different potential function or a

branching scheme, as is done in [15]. The question whether one can prove nonsatisfiability with local minima information remains open.

In light of the results presented here, the interior point approach can serve as an efficient tool in solving instances of SAT that before were considered unsolvable simply because SAT is NP-complete.

References

[1] C.E. Blair, R.G. Jeroslow and J.K. Lowe, Some results and experiments in programming techniques for propositional logic, Comp. Oper. Res. 5 (1986) 633–645.

[2] T.M. Cavalier, P.M. Pardalos and A.L. Soyster, Modeling and integer programming techniques applied to propositional calculus, Comp. Oper. Res., to appear.

[3] S.A. Cook, The complexity of theorem-proving procedures, in: *Proc. 3rd ACM Symp. on the Theory of Computing* (1971) pp. 151–158.

[4] M. Davis and H. Putnam, A computing procedure for quantification theory, J. ACM 7 (1960) 201–215.

[5] P.L. Hammer and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas* (Springer, 1968).

[6] P. Hansen, B. Jaumard and M. Minoux, Algorithm for deriving all logical conclusions implied by a set of Boolean inequalities, Math. Programming 34 (1968) 223–231.

[7] J.N. Hooker, A quantitative approach to logical inference, Decision Support Systems 4 (1988) 45–69.

[8] J.N. Hooker, Generalized resolution and cutting planes, Ann. Oper. Res. 12 (1988) 217–239.

[9] J.N. Hooker, Resolution vs. cutting plane solution of inference problems: Some computational experience, Oper. Res. Lett. 7 (1) (1988) 1–7.

[10] J.N. Hooker and C. Fedjki, Branch-and-cut solution of inference problems in propositional logic, Technical Report 77-88-89, GSIA, Carnegie Mellon University, Pittsburgh, PA 15213 (August 1989), Ann. Math. AI 1 (1990) 123–139.

[11] R.G. Jeroslow, Computation-oriented reductions of predicate to propositional logic, Decision Support Systems 4 (1988) 183–197.

[12] N. Karmarkar, An interior-point approach to NP-complete problems – extended abstract, in: *Mathematical Developments Arising from Linear Programming Algorithms*, Summer Research Conf. sponsored jointly by AMS, IMS and SIAM, Bowdoin College, Brunswick, Maine (June 1988).

[13] N. Karmarkar, M.G.C. Resende and K.G. Ramakrishnan, An interior-point algorithm for zero–one integer programming, in: *13th Int. Symp. on Mathematical Programming*, Mathematical Programming Society, Tokyo (September 1988).

[14] N. Karmarkar, M.G.C. Resende and K.G. Ramakrishnan, An interior-point algorithm to solve computationally difficult set covering problems, Technical report, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 (1989).

[15] N. Karmarkar, M.G.C. Resende and K.G. Ramakrishnan, An interior-point approach to the maximum independent set problem in dense random graphs, Technical report, Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 (1989).

[16] K. Levenberg, A method for the solution of certain problems in least squares, Quart. Appl. Math. 2 (1944) 164–168.

[17] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM J. Appl. Math. 11 (1963) 431–441.

[18] B.A. Murtagh and M.A. Saunders, MINOS 5.0 user's guide, Technical Report SOL 83-20, Dept. of Operations Research, Stanford University, Stanford, CA (1983).
[19] N.J. Nilson, *Principles of Artificial Intelligence* (Tioga, 1980).
[20] S. Sahni, Computationally related problems, SIAM J. Computing 3 (1974) 262–279.
[21] H.P. Williams, Logical problems and integer programming, Bull. Inst. Math. Appl. 13 (1977) 1820.
[22] H.P. Williams, Linear and integer programming applied to the propositional calculus, Systems Res. Inform. Sci. 2 (1987) 81–100.
[23] R.R. Yager, A mathematical programming approach to inference with the capability of implementing default rules, Int. J. Man–Machine Studies 29 (1988) 685–714.