Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

A linear-time transformation of linear inequalities into conjunctive normal form

J.P. Warners

Computer Science/Department of Software Technology

CS-R9631 1996

Report CS-R9631
ISSN 0169-118X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

A Linear–Time Transformation of Linear Inequalities into Conjunctive Normal Form

Joost P. Warners*

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

e–mail: joostw@cwi.nl

Abstract

We present a technique that transforms any binary programming problem with integral coefficients to a satisfiability problem of propositional logic in linear time. Preliminary computational experience using this transformation, shows that a pure logical solver can be a valuable tool for solving binary programming problems. In a number of cases it competes favourably with well known techniques from operations research, especially for hard unsatisfiable problems.

CR Subject Classification (1991): F.4: Mathematical Logic and Formal Languages; G.2: Discrete Mathematics.
AMS Subject Classification (1991): 03B05: Classical Propositional Logic; 90C10: Integer Programming.
Keywords & Phrases: Linear inequalities, Conjunctive Normal Form, Horn cardinality clauses.

1 Introduction

The satisfiability problem of propositional logic (SAT) is considered important in many disciplines, such as mathematical logic, electrical engineering, computer science and operations research. It is the original NP–complete problem (Cook, [3]). It is well known that any problem of propositional logic can be formulated as a binary programming problem [8]. In the past years, mathematical programming techniques such as branch and bound and branch and cut have been applied to solve the satisfiability problem, with some success [2, 10]. On the other hand, it is also possible to apply techniques from mathematical logic and computer science, such as resolution [12], to solve specific binary programming problems. However, efficiently transforming a binary programming problem to a satisfiability problem is generally not a trivial task. To our knowledge, until now, no transformations are known that transform an arbitrary binary programming

*This research was supported by the Dutch Organization for Scientific Research (SION/NWO) under grant 612-33-001.

problem to a satisfiability problem in linear time.

A transform that may yield exponentially many clauses was introduced by Granot and Hammer [5]. Barth [1] studied the transformation to so called *extended* or *Horn cardinality clauses* [9], which in the worst case also may yield an exponential number of extended clauses. These transformations do not require the introduction of new variables. Hooker [7] shows that to replace *Horn cardinality clauses*, when no new variables are introduced, an exponential number of classical clauses is required.

In this paper we present a technique that transforms any binary integer programming problem with integral coefficients to a satisfiability problem in linear time. This entails the introduction of a substantial, but linear in the length of the input, number of additional variables and clauses.

This paper is organized as follows. In the next section we introduce some notation. The third section deals with a transformation that may result in an exponential number of clauses. In the fourth section we describe the new transformation. For comparison, in the subsequent section we consider a special case for which a polynomial time transformation is already known. Finally, in Section 6 we report on some computational results.

2 Notation

We use propositional formulae in *conjunctive normal form* (CNF), or *clausal* form. A formula is a conjunction of *clauses*. Each clause is a disjunction of *literals*, each of which is an atomic proposition or its negation. We denote the atoms by letters, and the negation operator by $\neg$. Literals are connected by the binary disjunction operator $\vee$ to form a clause. Clauses are connected by the binary conjunction operator $\wedge$ to form a formula. So, each clause C_i is of the form

$$C_i = \bigvee_{j \in P_i} p_j \vee \bigvee_{j \in N_i} \neg p_j$$

where $P_i \subseteq \{1, \ldots, m\}$ and $N_i \subseteq \{1, \ldots, m\}$, $P_i \cap N_i = \emptyset$, are sets of indices. The conjunction of clauses, which we denote as ψ, is given by

$$\psi = \bigwedge_{i=1}^{n} C_i = \bigwedge_{i=1}^{n} (\bigvee_{j \in P_i} p_j \vee \bigvee_{j \in N_i} \neg p_j)$$

We will also use the binary operators $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence), that can be eliminated to obtain a CNF using the following rules (see e.g. Van Dalen [4]):

$$\begin{aligned} (p \leftrightarrow q) &\equiv (\neg p \vee q) \wedge (p \vee \neg q) \\ (p \rightarrow q) &\equiv (\neg p \vee q) \end{aligned}$$

By a *truth value assignment* we mean a mapping from the m-dimensional unit cube, where m is the number of different atoms, to $\{0, 1\}$. We associate the value '0' with *false* and the value '1'

with *true*. The SAT problem is to determine whether some assignment of truth values to atoms makes a given formula true. An example of a (satisfiable) formula is

$$\psi = (p_1 \vee \neg p_3) \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_2 \vee p_3).$$

We can formulate any clause C_i as a *linear inequality*. A linear inequality consists of a sum of terms, where each term is a product of a variable and a (positive or negative) coefficient, one of the relations $>$ (greater than), $\geq$ (greater than or equal to), $<$ (less than) or $\leq$ (less than or equal to), and a *right hand side value*. It is satisfied when the variables are given values such that the sum on the left hand side has the proper relation to the right hand side. For example, the linear inequality $5x_1 - 3x_2 + 8x_3 \leq 4$, is satisfied when we take $x_1 = x_2 = x_3 = 0$. We can write a general linear inequality in the form

$$\sum_{i=1}^{m} a_i x_i = a^T x \leq b,$$

where the second expression denotes the inner product of the vectors a and x, both of which consist of m elements.

In the following, a proposition letter p_i corresponds to a binary variable x_i. The clause C_i can now be formulated as a linear inequality in the following way [8].

$$\sum_{j \in P_i} x_j + \sum_{j \in N_i} (1 - x_j) \geq 1, \tag{1}$$

or equivalently

$$\sum_{j \in P_i} x_j - \sum_{j \in N_i} x_j \geq 1 - |N_i|.$$

By the notation $|N_i|$ we denote the *cardinality* of the set N_i, that is the number of elements that N_i contains. Obviously, (1) is satisfied if and only if at least one of the terms in its left hand side contributes a '1', i.e. at least one of the literals is true.

Finally, given a natural number a we define the set B_a to be the set containing all powers of two that contribute to the binary representation of a. For example, if $a = 19$ then $B_a = \{0, 1, 4\}$ since $19 = 2^0 + 2^1 + 2^4$.

3 An exponential transformation

We first review a transformation that may yield an exponential number of clauses (see Granot and Hammer [5]). We consider an inequality of the form

$$a^T x = \sum_{i=1}^{m} a_i x_i \leq b, \tag{2}$$

where we assume that the a_i's are positive integers. We can do this without loss of generality, since if for some i we have that a_i is negative, we can replace $a_i x_i$ by the positive term $-a_i(1-x_i)$

while adding $-a_i$ to the right hand side b. If we let $y_i = 1 - x_i$, then y_i is again a binary variable.

One way of transforming an inequality into a set of clauses is the following. We first give a definition (see e.g. Nemhauser and Wolsey [11]).

Definition 1 *A* minimal cover *of an inequality of the form (2) is a set* $MC \subseteq \{1, \ldots, m\}$ *for which for all* $j \in MC$

$$\sum_{i \in MC \setminus \{j\}} a_i \leq b < \sum_{i \in MC} a_i.$$

For a minimal cover MC of an inequality $a^T x \leq b$, it will be clear that if we set $x_i = 1$ for all $i \in MC$, the inequality will be violated. In other words, *not every* x_i, $i \in MC$, should be equal to one. If we associate a proposition letter p_i with each variable x_i, then we can express this with the following logical expression:

$$\neg \bigwedge_{i \in MC} p_i,$$

which is equivalent with the clause

$$\bigvee_{i \in MC} \neg p_i.$$

Now we can state the following theorem.

Theorem 1 *Given an inequality of the form (2). Let* $\mathcal{C}$ *be the set containing all its distinct minimal covers. Let CNF*($\mathcal{C}$) *denote the conjunction of clauses that can be constructed from the minimal covers in* $\mathcal{C}$*. Then a binary solution* x *satisfies the inequality if and only if its associated truth value assignment to the proposition letters* $p_1, \ldots, p_m$ *satisfies CNF*($\mathcal{C}$).

Proof:

- Given x such that $a^T x \leq b$. Suppose that x does not satisfy the clause $C \in \text{CNF}(\mathcal{C})$. This implies that

  $$\bigwedge_{i \in C} p_i$$

  is true, which implies that $x_i = 1$, for all $i \in C$. This implies that $a^T x > b$, so we arrive at a contradiction.

- Now we are given an x that satisfies $\text{CNF}(\mathcal{C})$. Let $\Omega = \{i \in \{1, \ldots, |\Omega|\} : x_i = 1\}$. Suppose that

  $$a^T x = \sum_{i \in \Omega} a_i x_i = \sum_{i \in \Omega} a_i > b.$$

  Sort the $i \in \Omega$ such that $a_{i_1} \geq a_{i_2} \geq \ldots \geq a_{i_{|\Omega|}}$. Now for some t it holds that

  $$\sum_{k=1}^{t} a_{i_k} \leq b,$$

  while

  $$\sum_{k=1}^{t+1} a_{i_k} > b.$$

We conclude that $\{i_1, i_2, \ldots, i_{t+1}\}$ is a minimal cover of $a^T x \leq b$, so by construction there must be a clause

$$\bigvee_{k=1}^{t+1} \neg p_k.$$

Again, we have arrived at a contradiction. □

Example: Consider the inequality

$$4x_1 + 6x_2 - 3x_3 - 5x_4 + 10x_5 \leq 7.$$

We rewrite this as

$$4x_1 + 6x_2 + 3(1 - x_3) + 5(1 - x_4) + 10x_5 \leq 15.$$

This is equivalent with the following set of clauses:

$$\begin{array}{llllllllll}
 & & \neg p_2 & & & & & \vee & \neg p_5 \\
\neg p_1 & & & & & \vee & p_4 & \vee & \neg p_5 \\
\neg p_1 & & & \vee & p_3 & & & \vee & \neg p_5 \\
 & & & & p_3 & \vee & p_4 & \vee & \neg p_5 \\
\neg p_1 & \vee & \neg p_2 & \vee & p_3 & \vee & p_4 & &
\end{array}$$

Observe that if a variable x_i has a *positive* coefficient a_i, its corresponding proposition letter p_i occurs only *negated*. If the coefficient a_i is *negative*, then the proposition letter corresponding to x_i occurs *unnegated*.

To conclude this section, consider the case in which all a_i are equal to one, i.e.

$$\sum_{i=1}^{m} x_i \leq b.$$

The number of distinct minimal covers for such an inequality is equal to

$$\left(\begin{array}{c} m \\ b+1 \end{array} \right).$$

If $b + 1 = \frac{m}{2}$ this number is $\mathcal{O}(2^m)$. This demonstrates that for certain inequalities, the number of clauses that is required to replace them is exponential in the length of the inequality.

4 A linear transformation

Suppose we are given an inequality of the form (2), i.e.

$$\sum_{i=1}^{m} a_i x_i \leq b$$

Again, we assume that all the a_i's are positive and integral numbers. Furthermore, we assume that the inequality is nontrivial, i.e.

$$\sum_{i=1}^{m} a_i > b.$$

We consider the binary representation of each a_i. Now let a_{max} be the maximal entry of the vector a, and let M be such that $2^{M-1} \leq a_{max} < 2^M$. In other words M is the maximum natural number such that

$$M \leq 1 +^2 \log(a_{max}).$$

Each of the a_i's can be written in its binary representation, i.e.

$$a_i = \sum_{k=0}^{M-1} a_k^{(i)} \cdot 2^k,$$

where the $a_k^{(i)}$'s are either zero or one. We associate a proposition letter $\overline{p}_k^{(i)}$ with each coefficient $a_k^{(i)}$, and transform a_i as follows.

$$trans(a_i) = \bigwedge_{k \in B_{a_i}} \overline{p}_k^{(i)} \wedge \bigwedge_{k \notin B_{a_i}} \neg \overline{p}_k^{(i)}. \tag{3}$$

Furthermore, we associate a proposition letter p_{x_i} with the (binary) variable x_i. Recall that B_{a_i} is the set of indices j for which $a_k^{(i)} = 1$.

We now give a formal description of the transformation. Subsequently, we will explain it in more detail. In the following I is a set of indices. We introduce the sets $\lfloor I \rfloor \subseteq I$ and $\lceil I \rceil \subset I$ for which the following holds:

$$\lfloor I \rfloor \cup \lceil I \rceil = I;\ \lfloor I \rfloor \cap \lceil I \rceil = \emptyset;\ |\lfloor I \rfloor| \geq |\lceil I \rceil|;\ |\lfloor I \rfloor| - |\lceil I \rceil| \leq 1.$$

Or in words, the sets $\lfloor I \rfloor$ and $\lceil I \rceil$ are a partition of I. We shall denote the proposition letters that represent the sum

$$\sum_{i \in I} a_i x_i,$$

by

$$\{p_k^{(I)}\}_{k=0,1,\ldots,M_I},$$

where $M_I = M + \log(|I|)$. In the following we shall omit the subscript $k = 0, 1, \ldots, M_I$ and assume that k runs between zero and an appropriate upper bound. The transformation of the sum of products $a_i x_i$ over the (nonempty) index set I is recursively given by

$$trans(\sum_{i \in I} a_i x_i) = \begin{cases} trans(\sum_{i \in \lfloor I \rfloor} a_i x_i) \wedge trans(\sum_{i \in \lceil I \rceil} a_i x_i) \wedge \mathcal{T}^+(\{p_k^{(I)}\}, \{p_k^{(\lfloor I \rfloor)}\}, \{p_k^{(\lceil I \rceil)}\}), & \text{if } |I| \geq 2 \\ trans(a_i) \wedge \mathcal{T}^*(\{p_k^{(i)}\}, \{\overline{p}_k^{(i)}\}, p_{x_i}), & \text{if } I = \{i\}. \end{cases}$$

The transformation operator $\mathcal{T}^+(\{p_k^{(U)}\},\{p_k^{(V)}\},\{p_k^{(W)}\})$ with $U = V \cup W$, V and W nonempty, is given by

$$\left(p_0^{(U)} \leftrightarrow (p_0^{(V)} \leftrightarrow \neg p_0^{(W)})\right) \wedge \tag{4}$$

$$\left(c_{01}^{(U)} \leftrightarrow (p_0^{(V)} \wedge p_0^{(W)})\right) \wedge \tag{5}$$

$$\bigwedge_{k=1,\ldots,M_U} \left(p_k^{(U)} \leftrightarrow (p_k^{(V)} \leftrightarrow p_k^{(W)} \leftrightarrow c_{k-1,k}^{(U)})\right) \wedge \tag{6}$$

$$\bigwedge_{k=1,\ldots,M_U-1} \left(c_{k,k+1}^{(U)} \leftrightarrow (p_k^{(V)} \wedge p_k^{(W)}) \vee (p_k^{(V)} \wedge c_{k-1,k}^{(U)}) \vee (p_k^{(V)} \wedge c_{k-1,k}^{(U)})\right) \wedge \tag{7}$$

$$\left(c_{M_U-1,M_U}^{(U)} \leftrightarrow p_{M_U}^{(U)}\right). \tag{8}$$

Furthermore, the transformation operator $\mathcal{T}^*(\cdot)$ is given by

$$\mathcal{T}^*(\{p_k^{(i)}\},\{\overline{p}_k^{(i)}\},p_{x_i}) = \bigwedge_{k=0}^{M_i} \left(p_k^{(i)} \leftrightarrow (\overline{p}_k^{(i)} \wedge p_{x_i})\right). \tag{9}$$

Now let $\mathcal{M} = \{1,\ldots,m\}$. To express that the right hand side value b may not be exceeded, we transform it in the following way.

$$\mathcal{T}(b) = \bigwedge_{k \notin B_b} \left(p_k^{(\mathcal{M})} \rightarrow \neg \bigwedge_{j \in B_b : j>k} p_j^{(\mathcal{M})} \right).$$

In the following we shall try to clarify the procedure and give the CNF form of the logical expressions introduced above.

First we consider the operator $\mathcal{T}^+(\{p_k^{(U)}\},\{p_k^{(V)}\},\{p_k^{(W)}\})$ that performs the *addition* of two numbers in binary notation. The following figure will help in understanding the procedure. Let $U = V \cup W$.

$$\begin{array}{cccccccc} & p_{M_U-1}^{(V)} & p_{M_U-2}^{(V)} & \cdots & p_2^{(V)} & p_1^{(V)} & p_0^{(V)} & \\ & p_{M_U-1}^{(W)} & p_{M_U-2}^{(W)} & \cdots & p_2^{(W)} & p_1^{(W)} & p_0^{(W)} & + \\ \hline p_{M_U}^{(U)} & p_{M_U-1}^{(U)} & p_{M_U-2}^{(U)} & \cdots & p_2^{(U)} & p_1^{(U)} & p_0^{(U)} & \end{array} \tag{10}$$

For example, we have that $p_0^{(U)}$ is *true* if and only if *exactly one* of the propositions $p_0^{(V)}$ and $p_0^{(W)}$ is true. This is expressed by (4), which expands to the following CNF.

$$\begin{array}{ccccc} p_0^{(U)} & \vee & p_0^{(V)} & \vee & \neg p_0^{(W)} \\ p_0^{(U)} & \vee & \neg p_0^{(V)} & \vee & p_0^{(W)} \\ \neg p_0^{(U)} & \vee & \neg p_0^{(V)} & \vee & \neg p_0^{(W)} \\ \neg p_0^{(U)} & \vee & p_0^{(V)} & \vee & p_0^{(W)} \end{array} \tag{11}$$

To obtain an expression for $p_1^{(U)}$ we first introduce a "carry"-proposition $c_{01}^{(U)}$. This is true, only if both $p_0^{(V)}$ and $p_0^{(W)}$ are true; see (5) which in turn we can write as three clauses:

$$\begin{array}{ccccc} c_{01}^{(U)} & \vee & \neg p_0^{(V)} & \vee & \neg p_0^{(W)} \\ \neg c_{01}^{(U)} & \vee & p_0^{(V)} & & \\ \neg c_{01}^{(U)} & & & \vee & p_0^{(W)} \end{array} \tag{12}$$

Now we have that $p_k^{(U)}$ evaluates to true if either one or three of the propositions $p_k^{(V)}, p_k^{(W)}$ and $c_{k-1,k}^{(U)}$ are true. This is expressed by (6), that has the following CNF.

$$\begin{array}{lllllll}
p_k^{(U)} & \vee & \neg p_k^{(V)} & \vee & p_k^{(W)} & \vee & c_{k-1,k}^{(U)} \\
p_k^{(U)} & \vee & \neg p_k^{(V)} & \vee & \neg p_k^{(W)} & \vee & \neg c_{k-1,k}^{(U)} \\
p_k^{(U)} & \vee & p_k^{(V)} & \vee & \neg p_k^{(W)} & \vee & c_{k-1,k}^{(U)} \\
p_k^{(U)} & \vee & p_k^{(V)} & \vee & p_k^{(W)} & \vee & \neg c_{k-1,k}^{(U)} \\
\neg p_k^{(U)} & \vee & p_k^{(V)} & \vee & p_k^{(W)} & \vee & c_{k-1,k}^{(U)} \\
\neg p_k^{(U)} & \vee & p_k^{(V)} & \vee & \neg p_k^{(W)} & \vee & \neg c_{k-1,k}^{(U)} \\
\neg p_k^{(U)} & \vee & \neg p_k^{(V)} & \vee & \neg p_k^{(W)} & \vee & c_{k-1,k}^{(U)} \\
\neg p_k^{(U)} & \vee & \neg p_k^{(V)} & \vee & p_k^{(W)} & \vee & \neg c_{k-1,k}^{(U)}
\end{array} \tag{13}$$

Subformulae (7) and (8) give the logical expressions for $c_{k,k+1}^{(U)}$, $k = 1, \ldots, M_U - 1$ and $p_{M_U}^{(U)}$ respectively. The CNF of the first is given below. Note that we may substitute $p_{M_U}^{(U)}$ for $c_{M_U-1,M_U}^{(U)}$, thus eliminating (8).

$$\begin{array}{lllllll}
c_{k,k+1}^{(U)} & \vee & \neg p_k^{(V)} & \vee & \neg p_k^{(W)} & & \\
c_{k,k+1}^{(U)} & \vee & \neg p_k^{(V)} & & & \vee & \neg c_{k-1,k}^{(U)} \\
c_{k,k+1}^{(U)} & & & \vee & \neg p_k^{(W)} & \vee & \neg c_{k-1,k}^{(U)} \\
\neg c_{k,k+1}^{(U)} & \vee & p_k^{(V)} & \vee & p_k^{(W)} & & \\
\neg c_{k,k+1}^{(U)} & \vee & p_k^{(V)} & & & \vee & c_{k-1,k}^{(U)} \\
\neg c_{k,k+1}^{(U)} & & & \vee & p_k^{(W)} & \vee & c_{k-1,k}^{(U)}
\end{array} \tag{14}$$

In the above we have been very accurate, and we may relax this accuracy in the following way. Consider again Figure (10). For example, we need for $p_0^{(U)}$ to be true, that either $p_0^{(V)}$ *or* $p_0^{(W)}$ is true, which is expressed by (4), i.e. we require *equivalence*. However, we can relax this to

$$(p_0^{(V)} \leftrightarrow \neg p_0^{(W)}) \rightarrow p_0^{(U)}, \tag{15}$$

as *implication* suffices. Observe that $p_0^{(U)}$ might be true when both $p_0^{(V)}$ and $p_0^{(W)}$ are false. This transformation is slightly less restrictive. We can also replace the equivalence by implication in expressions (5) to (8). This results in less clauses when transforming to conjunctive normal form (the number of additional clauses is, roughly speaking, halved). Consider the clauses in (11) to (14): all clauses beginning with a negated proposition letter are left out. In the following we will consider the transformation with equivalence. In the final section of this paper, we report on computational experience with both choices. Observe that the idea that is used here is similar to Wilson's extension [16] of the transformation by Blair et al. [2] to transform logical formulas to CNF.

Now we consider the operator $\mathcal{T}^*(\{p_k^{(i)}\}, \{\overline{p}_k^{(i)}\}, p_{x_i})$ that is in fact the *multiplication* of a number

in binary notation with a binary number, i.e.

$$\begin{array}{cccccc} \overline{p}^{(i)}_{M_i-1} & \overline{p}^{(i)}_{M_i-2} & \cdots & \overline{p}^{(i)}_2 & \overline{p}^{(i)}_1 & \overline{p}^{(i)}_0 \\ & & & & & p_{x_i} \quad \times \\ \hline p^{(i)}_{M_i} & p^{(i)}_{M_i-1} & p^{(i)}_{M_i-2} & \cdots & p^{(i)}_2 & p^{(i)}_1 & p^{(i)}_0 \end{array} \tag{16}$$

So we have that the proposition $p^{(i)}_k$ is true if and only if both $\overline{p}^{(i)}_k$ and p_{x_i} are true, which is expressed by (9). Note that, using unit resolution, expression (9) in conjunction with (3) reduces to

$$\bigwedge_{k \in B_{a_i}} \left(p^{(i)}_k \leftrightarrow p_{x_i} \right) \wedge \bigwedge_{k \notin B_{a_i}} \neg p^{(i)}_k,$$

which has the CNF

$$\begin{array}{l} (\neg p^{(i)}_k \vee p_{x_i}) \wedge (p^{(i)}_k \vee \neg p_{x_i}), \text{ for all } k \in B^{(i)}_a \\ \neg p^{(i)}_k, \text{ for all } k \notin B^{(i)}_a. \end{array}$$

Finally, the CNF of $\mathcal{T}(b)$ is given by

$$\bigwedge_{k \notin B_b} \left(\neg p^{(\mathcal{M})}_k \vee \bigvee_{j \in B_b : j > k} \neg p^{(\mathcal{M})}_j \right).$$

Complexity To get an indication of the complexity of the procedure, we shall make use of the following equality.

$$\sum_{i=1}^{r} i 2^{-i} + r 2^{-r} + 2^{1-r} = 2, \qquad r = 1, 2, \ldots \tag{17}$$

For completeness, we prove this equality.

Proof: For $r = 1$ we have that $1 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} + 1 = 2$. Suppose the equality holds for r. Using induction, we have to prove that it also holds for $r + 1$:

$$\begin{aligned} \sum_{i=1}^{r+1} i 2^{-i} + (r+1) 2^{-r-1} + 2^{-r} &= \sum_{i=1}^{r} i 2^{-i} + 2(r+1) 2^{-r-1} + 2^{-r} \\ &= \sum_{i=1}^{r} i 2^{-i} + r 2^{-r} + 2 \cdot 2^{-r} \\ &= 2 \end{aligned}$$

□

The number of operations that is required to perform the transformation is of the order of magnitude of

$$\frac{m}{2}(M+1) + \frac{m}{4}(M+2) + \ldots + (M + \log(m)).$$

If we take r such that $2^{r-1} \leq m < 2^r$, the above is bounded by (using (17)),

$$2^r \sum_{i=1}^{r} 2^{-i}(M+i) = (2^r - 1)M + 2^{r+1} - r + 2$$

which is of the order $\mathcal{O}(mM) = \mathcal{O}(m\log(a_{max}))$. We conclude that the transformation requires linear time, assuming that a_{max} is a priori bounded.

Let us look somewhat more closely at how many variables and clauses we need to introduce to perform the transformation. To compute this, note that adding two terms of which the largest is represented by N proposition letters, we introduce $2N$ new variables and at most $14N - 7$ new clauses. Furthermore, if $m = 2^r$ for some natural number r we can easily compute the required number of variables and clauses, using (17). Let us denote the number of additional variables by var and the number of additional clauses by cl

$$var_{2^r} = \sum_{i=1}^{r} 2^{r+1-i}(M + i - 1) = 2(2^r(M + 1) - (M + r + 1)) \tag{18}$$

$$cl_{2^r} = \sum_{i=1}^{r} 2^{r-i}(14(M + i) - 21) = 7(2^r(2M + 1) - 2(M + r) - 1). \tag{19}$$

To compute the number of variables required to transform an inequality of length m, let B_m be the set of powers of 2 that are in the binary representation of m. We make use of the following recursive formula:

$$var_m = var_{2^k} + var_{m-2^k} + 2l(var_{2^k}), \tag{20}$$

where $k = \max(B_m)$ and $l(var_{2^k})$ denotes the length of the binary representation of var_{2^k}. It is easily understood that $l(var_{2^k}) > l(var_{m-2^k})$. By construction, we have that $l(var_{2^k}) = M + k$. Substituting this and (18) in (20) we obtain

$$var_m = 2^{k+2} + 2^{k+1}(M - 1) - 2 + var_{m-2^k}.$$

We have that $B_{m-2^k} = B_m \backslash \{k\}$. Using this we can derive the following upper bound for var_m:

$$var_m \leq 2[m(M + 1) - (M + |B_m| + \min(B_m)] \ \leq \ 2m(2 + \log(a_{max})).$$

In the same manner we can derive an upper bound on the number of clauses:

$$cl_m \leq 7[m(2M + 1) - 2(M + |B_m| + min(B_m)) + 1] + M_{\mathcal{M}} - |B_b| \ \leq \ 7m(3 + 2\log(a_{max})),$$

where the term $M_{\mathcal{M}} - |B_b|$ is the number of clauses required to process the right hand side. Note that we have introduced a number of redundant variables, as we can straightforwardly substitute p_{x_i} for any $p_k^{(i)}$, $k \in B_{a_i}$. These variables are not included in the last computations.

Remarks

- The general transformation described above entails, in most practical situations, the introduction of a number of equivalent variables and clauses. There are a number of refinements that make it possible to reduce the number of additional variables and clauses, which we briefly mention. In many cases these redundant variables and clauses will be recognized immediately by the logical solver used, but there may be situations in which it is beneficial to remove them beforehand.

- As mentioned in the previous section, we do not need to introduce any new variables and clauses to replace single terms.
- Consider the sum (10). If the $p_k^{(i)}$, $k \in B_{a_i}$, and $p_k^{(j)}$, $k \in B_{a_j}$, all refer to a single variable x_i resp. x_j we do not need to introduce any "carry"-variables and also some of the "sum"-variables will be equivalent. Note that in the general case we can also leave out the carry-variables, but this will lead to introducing more and longer clauses.
- By keeping track of the maximal value the sum of two terms can take, we can in some cases see beforehand that certain proposition letters must get the value false, so we need not introduce them.
- We can divide all coefficients and the hand side by their greatest common divider, so as to reduce the number of new variables required.

- System of linear inequalities can be transformed to CNF by applying the procedure described above to each of the inequalities separately. First, each equality must be brought to the form (2). Note that it is important to keep track of negative coefficients: if a variable x_i had a negative coefficient, its associated proposition letter p_{x_i} must be negated in (9) c.q. (17).
- An inequality that has rational coefficients can also be transformed by our procedure. First, its coefficients must be multiplied by an appropriate number to obtain an inequality with integral coefficients. For example, one could take the product of the denominators of the coefficients.

Example: Consider the inequality

$$2x_1 + 4x_2 + 5x_3 \leq 6. \tag{21}$$

We transform (21) by first taking the sum of the first two terms; this requires no additional variables and clauses. Subsequently we add the third term. The CNF we obtain is given below, where $U = \{1, 2, 3\}$. Note that modelling the right hand side results in (among others) a single literal clause; by applying unit resolution we can directly reduce the size of the CNF from 9 to 6 clauses.

$$\begin{array}{lllllllllllllllll}
\neg p_2^{(U)} & \vee & \neg p_{x_2} & \vee & \neg p_{x_3} & \quad \neg p_3^{(U)} & \vee & p_{x_2} & & & \neg p_3^{(U)} & & & & \\
\neg p_2^{(U)} & \vee & p_{x_2} & \vee & p_{x_3} & \quad \neg p_3^{(U)} & & & \vee & p_{x_3} & & & & & \\
p_2^{(U)} & \vee & p_{x_2} & \vee & \neg p_{x_3} & \quad p_3^{(U)} & \vee & \neg p_{x_2} & \vee & \neg p_{x_3} & \neg p_2^{(U)} & \vee & \neg p_{x_1} & \vee & \neg p_{x_3} \\
p_2^{(U)} & \vee & \neg p_{x_2} & \vee & p_{x_3} & & & & & & & & & &
\end{array}$$

This example is of course very nice in the sense that we need to introduce only a small number of new variables. Note that, using minimal covers, (21) is equivalent to the following CNF:

$$(\neg p_{x_1} \vee \neg p_{x_3}) \wedge (\neg p_{x_2} \vee \neg p_{x_3})$$

Example: [11, 1] Consider the inequality

$$300x_1 + 300x_4 + 285x_5 + 285x_6 + 265x_8 + 265x_9 + 230x_{12} + 230x_{13} + 190x_{14} + 200x_{22} + 400x_{23} + 200x_{24} + 400x_{25} + 200x_{26} + 400x_{27} + 200x_{28} + 400x_{29} + 200x_{30} + 400x_{31} \leq 2700$$

The CNF that is equivalent to this inequality consists of 117520 clauses, when introducing no additional variables. When using extended clauses, 4282 of those are required [1]. The linear time transformation requires 165 variables and 895 clauses, and if the inequality is first divided by the greatest common divisor (5), this number reduces to 122 and 669 respectively.

5 Horn cardinality clauses

In this section we will consider a special class of linear inequalities, the *Horn cardinality clauses*, which have the form

$$\sum_{i=1}^{m} x_i \geq b. \tag{22}$$

This is the only form of inequalities that we are aware of, for which there exists a polynomial CNF expansion (Hooker [7]). The CNF equivalent of (22) is

$$\neg z_{ik} \vee p_{x_i}, \qquad i = 1, \ldots, m, k = 1, \ldots, b, \tag{23}$$

$$\bigvee_{i=1}^{m} z_{ik}, \qquad k = 1, \ldots, b, \tag{24}$$

$$\neg z_{ik} \vee \neg z_{jk}, \qquad i, j = 1, \ldots, n, i \neq j, k = 1, \ldots, b. \tag{25}$$

Here (23) says that x_i is true if some z_{ik} is true, and (24) combined with (25) say that for each k exactly one z_{ik} must be true.

The number of additional variables required to perform this transformation is bm and the number of clauses required is $\frac{1}{2}b(m^2 + m + 2)$.

If we use the transform of the previous section, we have $M = 1$ and with the additional insight that

$$var_m = var_{2^k} + var_{m-2^k} + 2k - 1,$$

where again $k = \max(B_m)$ and we use that $l(var_m) = l(var_{2^k})$, we find an expression for the number of additional variables:

$$var_m = 4m - 3|B_m| - 2\min(B_m) - 1.$$

Similarly, we obtain an expression for the number of clauses:

$$cl_m = 21m - 18|B_m| - 7(\max(B_m) + \min(B_m)) - 3.$$

Here we do not take into account the number of clauses that is required to represent the right hand side $m - b$ (note that we first have transformed (22) to a '$\leq$' inequality). This requires

another $1 + \max(B_m) - |B_b|$ clauses.

Comparing the two polynomial transforms, we observe that the number of additional variables and clauses in Hooker's transform is very much dependent on the right hand side b, whereas in our transform the right hand side only (slightly) influences the number of additional clauses. The application of the next section shows that, in specific cases, this can be considered as beneficial.

6 Computational experience

In this section we report on some computational experience with the linear transformation. Many combinatorial problems are *almost* satisfiability problems, in the sense that most of the constraints can be regarded as clauses; only a small number of constraints is different. Our transformation makes it possible to solve these problems with a logical solver. The aim of this section is to show that, given a particular combinatorial problem, a logical solver can be as efficient or more efficient than a mathematical programming package.

We consider the *Frequency Assignment Problem* (FAP): Given a set of *radio links* L, a set of *frequencies* F and a set if *interference constraints* D, assign to each radio link a frequency, such that the interference constraints are not violated, and the number of different frequencies used is below a certain maximum number (provided by the user). *Interference* is a phenomenon that occurs when two radio links that are situated near each other, get the same or close frequencies. An interference constraint is a triple (l, k, d_{lk}), where $d_{lk} \geq 0$ is the minimum distance required (in mHz) between the frequencies assigned to radio links l and k.

In Warners et al. [15] various mathematical models for this problem are developed. Here we use the model that has a structure similar to that of the *pigeon hole principle*, a notorious problem form mathematical logic which was used to prove the exponentiality of resolution [6]. We introduce the proposition letters p_{lf} and q_f:

$$p_{lf} = \begin{cases} \mathit{true} & \text{if frequency } f \text{ is assigned to radio link } l, \\ \mathit{false} & \text{otherwise.} \end{cases} \qquad \text{for all } l \in L,\ f \in F$$

$$q_f = \begin{cases} \mathit{true} & \text{if frequency } f \text{ is } \textit{not} \text{ assigned to any radio link,} \\ \mathit{false} & \text{otherwise.} \end{cases} \qquad \text{for all } f \in F$$

We associate a binary variable x_{lf} with each letter p_{lf} and a binary variable z_f with each letter q_f (i.e. given some $f \in F$, we have that $z_f = 1$ if and only if q_f is *true*). Let F_{min} be the minimal number of frequencies *not* to be used, then we can express the (FAP) as follows:

$$\bigvee_{f \in F} p_{lf}, \qquad \text{for all } l \in L \tag{26}$$

$$\neg p_{lf} \vee \neg p_{kg}, \qquad \text{for all } (f, g) \text{ such that } |f - g| \leq d_{lk} \tag{27}$$

$$\neg p_{lf} \vee \neg q_f, \qquad \text{for all } l \in L,\ f \in F, \tag{28}$$

with the additional constraint that

$$\text{at least } F_{min} \text{ propositions } q_f \text{ must be } \mathit{true}. \tag{29}$$

Here, (26) expresses that to each radio link a frequency must be assigned and (27) model the interference constraints. The clauses (28) keep track of which frequencies are assigned to at least one link, while (29) makes sure that not too many different frequencies are assigned. Only the last constraint is not in CNF. Since it can be written as

$$\sum_{f \in F} z_f \geq F_{min}, \tag{30}$$

we can straightforwardly transform it to CNF by our procedure (note that (30) is in fact a Horn cardinality clause).

We have selected a number of FAPs that are structured as described in Warners [14]. For these FAPs, the cardinality of F is typically equal to 24, so the number of additional variables and clauses to transform it are equal to 83 and 416 (when using the transformation with equivalence; when restricting to implication the number of additional clauses reduces to 192). Observe that for Hooker's transform, the number of additional clauses and variables are $301F_{min}$ resp. $24F_{min}$, which for $F_{min} > 10$ is substantially larger.

The selected problems were solved with the logical solver HeerHugo, developed by Jan Friso Groote at the CWI in Amsterdam, The Netherlands, and with the well known optimization package CPLEX. The tests were run on a HP9000/720, 100 MHz. Obviously, when using CPLEX we do not need to transform (30), while all clauses (26-28) can readily be written as linear inequalities (see the introduction). In the Tables 1–4 the results are summarized. The problems are solved for different values of F_{min}: the values of F_{min} are chosen around its optimal value (i.e. the maximal number of different frequencies not used). As CPLEX allows for an objective function, the problems are also solved with the objective to maximize F_{min} (under the constraints (26-28)); see column 4. Furthermore, results are reported of both using the transformation to CNF with implication ('$\rightarrow$') and equivalence ('$\leftrightarrow$'). Times are given in seconds. Numbers of variables and constraints c.q. clauses do not include the additional variables and clauses due to the transformation.

Based on these experiments, we observe the following:

- For some of the larger problems, HeerHugo outperforms CPLEX, especially in the cases where the problem under consideration is just unsatisfiable due to the value of F_{min}. In a number of cases HeerHugo also performs better on satisfiable problems.
- The transformation to CNF with implication instead of equivalence, generally gives better results when the problem is satisfiable. This is most probably due to the fact that the transformation with implication allows more satisfiable assignments (if any).

We conclude from our experiments that a logical solver can be an efficient tool to solve difficult binary programming problems. We observe, however, that the results obtained by CPLEX can be improved by using a tighter linear model, and adding strong valid inequalities. See for an overview of efficient algorithms for the FAP, Tiourine et al. [13].

G10.6		solvers			
F_{min}	SAT ?	HeerHugo $\rightarrow$	HeerHugo $\leftrightarrow$	CPLEX feas.	CPLEX + obj.
19	SAT	1	14	.3	
20	SAT	13	10	.4	
21	SAT	1	10	.6	34
22	UNSAT	73	75	24	
23	UNSAT	2	2	2	

Table 1: G10.6: 158 variables, 1517 constraints

G16.6		solvers			
F_{min}	SAT ?	HeerHugo $\rightarrow$	HeerHugo $\leftrightarrow$	CPLEX feas.	CPLEX + obj.
19	SAT	22	24	4	
20	SAT	15	63	30	
21	SAT	15	40	15	71
22	UNSAT	154	193	173	
23	UNSAT	4	4	9	

Table 2: G16.6: 232 variables, 3225 constraints

G20.10		solvers			
F_{min}	SAT ?	HeerHugo $\rightarrow$	HeerHugo $\leftrightarrow$	CPLEX feas.	CPLEX + obj.
17	SAT	56	33	789	
18	SAT	845	781	709	
19	SAT	836	912	35039	103885
20	UNSAT	1495	1531	> 120000	
21	UNSAT	554	552	22941	

Table 3: G20.10: 296 variables, 7500 constraints

G20.6		solvers			
F_{min}	SAT ?	HeerHugo $\rightarrow$	HeerHugo $\leftrightarrow$	CPLEX feas.	CPLEX + obj.
19	SAT	191	120	9	
20	SAT	131	127	11	
21	SAT	124	116	12	2475
22	UNSAT	1233	1295	24378	
23	UNSAT	14	14	368	

Table 4: G20.6: 411 variables, 8979 constraints

References

[1] P. Barth. Linear 0–1 inequalities and extended clauses. Technical Report MPI–I–94–216, Max–Planck–Institut für Informatik, Im Stadtwald, D66123 Saarbrücken, Germany, 1994.

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

[3] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd annual ACM symposium on the Theory of Computing*, pages 151–158, 1971.

[4] D. Van Dalen. *Logic and structure*. Springer–Verlag, Berlin, 3rd edition, 1994.

[5] F. Granot and P.L. Hammer. On the use of boolean functions in 0-1 programming. *Methods of Operations Research*, 12:154–184, 1971.

[6] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.

[7] J.N. Hooker. Unpublished note.

[8] J.N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12:217–239, 1988.

[9] J.N. Hooker. Generalized resolution for 0–1 linear inequalities. *Annals of Mathematics and Artificial Intelligence*, 6:271–286, 1992.

[10] J.N. Hooker and C. Fedjki. Branch–and–cut solution of inference problems in propositional logic. *Annals of Mathematics and Artificial Intelligence*, 1:123–139, 1990.

[11] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[12] J.A. Robinson. A machine–oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[13] S. Tiourine, C. Hurkens, and J.K. Lenstra. An overview of algorithmic approaches to frequency assignment problems. Technical report, CALMA project, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1995.

[14] J.P. Warners. A potential reduction approach to the Radio Link Frequency Assignment Problem. Master's thesis, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995.

[15] J.P. Warners, T. Terlaky, C. Roos, and B. Jansen. A potential reduction approach to the frequency assignment problem. Technical Report 95-98, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, The Netherlands, 1995. Submitted for publication.

[16] J.M. Wilson. Compact normal forms in propositional logic and integer programming formulations. *Computers and Operations Research*, 17(3):309–314, 1990.