ELSEVIER

Information Processing Letters 68 (1998) 63–69

Information Processing Letters

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

Joost P. Warners [a,b,1]

[a] *CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*
[b] *Department of Technical Mathematics and Informatics, Faculty of Information Technology and Systems Delft University of Technology, P.O.Box 5031, 2600 GA Delft, The Netherlands*

Received 30 June 1998; received in revised form 31 August 1998
Communicated by P.M.B. Vitányi

Abstract

We present a technique that transforms a binary linear 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 specific binary linear programming problems, namely those that are in a sense 'almost' satisfiability problems. In a number of cases it competes favorably with well-known techniques from operations research.

Keywords: Satisfiability; Combinatorial optimization; Automatic theorem proving

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 was the first problem shown to be NP-complete (Cook [2]). It is well known that a problem of propositional logic can be reformulated as a formula in conjunctive normal form (CNF), in linear time [1]. In turn, a CNF formula can be directly expressed as a linear programming problem with binary variables (BLP) [4], i.e., as finding the optimum of a linear (or constant) objective function subject to linear constraints and the additional constraint that all variables must be binary. Thus, mathematical programming techniques such as branch and bound, and branch and cut have been applied to solve the satisfiability problem, with some success [1,6].

Conversely, techniques from logic might provide an effective complementary approach for solving BLPs (see also [5]). However, efficiently transforming a BLP to a CNF formula in general does not seem to be a trivial task. Note that from the theory of NP-completeness follows that a polynomial-time transformation must exist. Still, to the best of our knowledge, in the literature no transformations can be found that transform an arbitrary BLP to a CNF formula in linear time. Hooker shows a polynomial transformation not requiring the introduction of additional variables does not exist [3]; he gives a transformation that introduces a quadratic number of additional variables and clauses.

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

[1] Email: Joost.Warners@cwi.nl.

0020-0190/98/$ – see front matter
PII: S0020-0190(98)00144-6

The aim of this paper is two-fold. Firstly, we present an essentially simple technique to transform a BLP with integral coefficients to a CNF formula and show that it requires linear time. Secondly, we show that using this technique, specific BLPs can be solved efficiently by a logical solver. As an example we consider the frequency assignment problem.

This paper is organized as follows. In the next section we introduce some notation. In Section 3 the linear-time transformation is described. In the final section we report on some computational results.

2. Preliminaries

We use propositional formulas in conjunctive normal form (CNF). A CNF formula is a conjunction of clauses. Each clause is a disjunction of literals, each of which is an atomic proposition p_i or its negation $\neg p_i$. Literals are connected by the binary disjunction operator $\vee$ to form a clause. Clauses C_i are connected by the binary conjunction operator $\wedge$ to form a formula Φ. Thus

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

where $P_i, N_i \subseteq \{1, \ldots, m\}$, $P_i \cap N_i = \emptyset$. We will also use the binary operators $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence), that can be eliminated to obtain CNFs using the well known De Morgan's laws. A truth value assignment is a mapping from $\{0, 1\}^m$, where m is the number of different atoms, to $\{0, 1\}$. We associate the value '0' ('1') with *false* (*true*). The SAT problem is to determine whether some assignment of truth values to the atoms makes a formula true.

A linear inequality concerning the binary vector $x \in \{0, 1\}^m$ can be expressed as

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

where $a_i, b \in \mathbb{R}$. Note that without loss of generality the a_i may be assumed to be positive; in the following we assume the a_i to be positive natural numbers. Associating binary variables x_i with propositions p_i, clauses can be formulated as single linear inequalities [4]. Formulating linear inequalities as sets of clauses is the subject of the next section.

For a set P, $|P|$ denotes the cardinality of the set P, that is the number of elements contained in P. Furthermore, for a natural number a_i we define the set B_{a_i} to be the set containing all powers of two that contribute to the binary representation of a_i. For example, if $a_i = 19$ then $B_{a_i} = \{0, 1, 4\}$. Finally, $a_{\max}$ denotes the maximal entry of the vector a, and the natural number M is chosen such that $2^{M-1} \leqslant a_{\max} < 2^M$, i.e., M is the largest natural number such that $M \leqslant 1 + {}^2\log(a_{\max})$. Assuming that $a_{\max}$ is *a priori* bounded, M may be treated as a constant.

3. Transforming inequalities to CNF

Let us first give a brief outline of the idea of the transformation, which is essentially quite simple. On the lowest level, propositions and clauses are added to represent the single terms $a_i x_i$, using the binary representations of the a_i. The required logical expressions are derived from binary multiplication, and their CNF translations are added to the set of clauses; the expressions are such that the propositions associated with the term $a_i x_i$ are true if and only if $x_i = 1$. Then, on the next level, propositions are added to represent the sum (in binary notation) of disjoint pairs of terms, and logical expressions derived from binary addition are added to the set of clauses. A newly introduced proposition is true if and only if the corresponding bit is equal to 1 under the current valuation of the x_i. On the subsequent level sums of, again disjoint, *pairs of pairs* of terms are represented in a similar way. This process is repeated until the top level is reached where a set of propositions is added that represents the sum, in binary notation, over all the terms of the inequality under consideration. During the process the logical expressions on a certain level operate exclusively on the propositions introduced on that level and those introduced one level lower. Finally, logical expressions that operate on the top level propositions are derived to ensure that the right hand side b of the inequality is not exceeded. Here use is made of the binary representation of b. For example, let $b = 26$ and denote the top level proposition corresponding to bit $i + 1$ (i.e., the bit that contributes 2^i to the sum) by p_i. It holds that $B_b = \{1, 3, 4\}$ and it follows that $\neg p_i$ for $i \geqslant 5$, $p_2 \rightarrow \neg(p_3 \wedge p_4)$ and $p_0 \rightarrow \neg(p_1 \wedge p_3 \wedge p_4)$.

The resulting formula is satisfiable if and only if the original inequality allows a feasible binary solution. A satisfiable assignment restricted to the propositions corresponding to the original x_i variables yields a feasible binary solution of the inequality.

Below a formal recursive definition of the transformation is given. Subsequently, it is discussed in some more detail and it is shown that the transformation requires linear time. We associate propositions p_{x_i} with the (binary) variables x_i, $i = 1, \ldots, m$, and introduce propositions $\overline{p}_k^{(i)}$ for all $k = 1, \ldots, M$, $i = 1, \ldots, m$. In the following I is a set of indices. We use the sets $\lfloor I \rfloor \subseteq I$ and $\lceil I \rceil \subset I$ for which the following hold:

$$\lfloor I \rfloor \cup \lceil I \rceil = I; \quad \lfloor I \rfloor \cap \lceil I \rceil = \emptyset;$$
$$\big|\lfloor I \rfloor\big| \geqslant \big|\lceil I \rceil\big|; \quad \big|\lfloor I \rfloor\big| - \big|\lceil I \rceil\big| \leqslant 1.$$

The sets $\lfloor I \rfloor$ and $\lceil I \rceil$ are a partition of I. The propositions representing the sum $\sum_{i \in I} a_i x_i$ are denoted by $\{p_k^{(I)}\}$ where $k = 0, 1, \ldots, M_I$, with $M_I = M + \log(|I|)$.

The transform of a linear inequality of the form (1) is given by

$$\mathit{trans}\left(\sum_{i=1}^{m} a_i x_i \leqslant b\right) = \mathit{trans}\left(\sum_{i=1}^{m} a_i x_i\right) \wedge \mathit{trans}(\leqslant b).$$

The transformation of the sum of $a_i x_i$ over the index set $I \neq \emptyset$ is recursively defined as

$$\mathit{trans}\left(\sum_{i \in I} a_i x_i\right) = \begin{cases} \mathit{trans}\left(\sum_{i \in \lfloor I \rfloor} a_i x_i\right) \wedge \mathit{trans}\left(\sum_{i \in \lceil I \rceil} a_i x_i\right) \\ \quad \wedge \mathcal{T}^+\big(\{p_k^{(I)}\}, \{p_k^{(\lfloor I \rfloor)}\}, \{p_k^{(\lceil I \rceil)}\}\big), & \text{if } |I| > 1, \\ \mathit{trans}(a_i) \wedge \mathcal{T}^*\big(\{p_k^{(i)}\}, \{\overline{p}_k^{(i)}\}, p_{x_i}\big), & \text{if } I = \{i\}. \end{cases}$$

Here the operator $\mathcal{T}^+(\{p_k^{(U)}\}, \{p_k^{(V)}\}, \{p_k^{(W)}\})$ performs the *addition* of two numbers in binary notation $p_k^{(V)}$ and $p_k^{(W)}$, yielding the sum $p_k^{(U)}$. The operator $\mathcal{T}^*(\{p_k^{(i)}\}, \{\overline{p}_k^{(i)}\}, p_{x_i})$ performs the *multiplication* of a number in binary notation $\overline{p}_k^{(i)}$ with a binary number p_{x_i}. Later we give explicit expressions for these operators. Furthermore, we have that

$$\mathit{trans}(a_i) = \bigwedge_{k \in B_{a_i}} \overline{p}_k^{(i)} \wedge \bigwedge_{k \notin B_{a_i}} \neg \overline{p}_k^{(i)}. \tag{2}$$

Finally, let us define $\mathcal{M} = \{1, \ldots, m\}$. To process the right hand side b, we define

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

We have the following theorem.

Theorem 1. *Assuming $a_{\max}$ is a priori bounded, the above introduced transformation requires a number of operations that is linear in the length of the inequality.*

To prove the theorem, let us take a closer look at the number of additional variables and clauses we need to introduce to perform the transformation. To this end, we first specify the number of additional variables and clauses the binary multiplication and addition require. The following figure explains how to obtain the transformation operator $\mathcal{T}^+(\{p_k^{(U)}\}, \{p_k^{(V)}\}, \{p_k^{(W)}\})$. Let $U = V \cup W$.

$$\begin{array}{cccccc} & p_{M_U-1}^{(V)} & p_{M_U-2}^{(V)} & \cdots & p_2^{(V)} & p_1^{(V)} & p_0^{(V)} \\ & p_{M_U-1}^{(W)} & p_{M_U-2}^{(W)} & \cdots & p_2^{(W)} & p_1^{(W)} & p_0^{(W)} \ + \\ \hline p_{M_U}^{(U)} & p_{M_U-1}^{(U)} & p_{M_U-2}^{(U)} & \cdots & p_2^{(U)} & p_1^{(U)} & p_0^{(U)} \end{array} \tag{3}$$

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 expressed by (4). Thus transformation operator $\mathcal{T}^+(\{p_k^{(U)}\}, \{p_k^{(V)}\}, \{p_k^{(W)}\})$ with $U = V \cup W$, V and W nonempty, can be expressed as (using *carry* propositions)

$$\big(p_0^{(U)} \leftrightarrow \big(p_0^{(V)} \leftrightarrow \neg p_0^{(W)}\big)\big) \wedge \tag{4}$$
$$\big(c_{01}^{(U)} \leftrightarrow \big(p_0^{(V)} \wedge p_0^{(W)}\big)\big) \wedge \tag{5}$$
$$\bigwedge_{k=1,\ldots,M_U} \big(p_k^{(U)} \leftrightarrow \big(p_k^{(V)} \leftrightarrow p_k^{(W)} \leftrightarrow c_{k-1,k}^{(U)}\big)\big) \wedge \tag{6}$$
$$\bigwedge_{k=1,\ldots,M_U-1} \big(c_{k,k+1}^{(U)} \leftrightarrow \big(p_k^{(V)} \wedge p_k^{(W)}\big) \vee \big(p_k^{(V)} \wedge c_{k-1,k}^{(U)}\big) \vee \big(p_k^{(V)} \wedge c_{k-1,k}^{(U)}\big)\big) \wedge \tag{7}$$
$$\big(c_{M_U-1,M_U}^{(U)} \leftrightarrow p_{M_U}^{(U)}\big). \tag{8}$$

The logical expressions (4)–(8) can be readily rewritten into CNF [8]; taking $N = \max\{|V|, |W|\}$, $2N$ new variables and at most $14N - 7$ clauses are introduced.

The transformation operator $\mathcal{T}^*(\cdot)$ is obtained in a similar way, yielding

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

Note that, using unit resolution, this expression in conjunction with (2) reduces to

$$\bigwedge_{k \in B_{a_i}} \big(p_k^{(i)} \leftrightarrow p_{x_i}\big) \wedge \bigwedge_{k \notin B_{a_i}} \neg p_k^{(i)}. \tag{10}$$

From this we see that the propositions $p_k^{(i)}$, $k \in B_{a_i}$, can be eliminated by substituting them by p_{x_i}. Thus in the computations below these are not counted. Now let us denote by var_m and cl_m the numbers of additional variables and clauses to transform an inequality of length m.

Lemma 2. *We have the following upper bounds on var_m and cl_m.*

$$var_m \leqslant 2m\big(1 + \log(a_{\max})\big),$$
$$cl_m \leqslant 8m\big(1 + 2\log(a_{\max})\big).$$

Proof. If $m = 2^r$ for some natural number r the required number of variables and clauses can easily be computed:

$$var_{2^r} = \sum_{i=1}^{r} 2^{r-i}\big(2(M + i - 1)\big) = 2\big(2^r(M+1) - (M + r + 1)\big), \tag{11}$$

$$cl_{2^r} = \sum_{i=1}^{r} 2^{r-i}\big(14(M + i - 1) - 7\big) = 7\big(2^r(2M+1) - 2(M + r) - 1\big). \tag{12}$$

To compute var_m for arbitrary m, we use the following recursive formula:

$$var_m = var_{2^k} + var_{m-2^k} + 2\ell(var_{2^k}), \tag{13}$$

where $k = \max(B_m)$ and $\ell(var_{2^k})$ denotes the length of the binary representation of var_{2^k}. By construction, we have that $\ell(var_{2^k}) = M + k$. Substituting this and (11) in (13) we obtain

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

Using that $B_{m-2^k} = B_m \backslash \{k\}$, we derive the following upper bound on var_m:

$$var_m \leqslant 2\big[m(M+1) - \big(M + |B_m| + \min(B_m)\big)\big].$$

In a similar manner an upper bound on the number of clauses can be derived:

$$cl_m \leqslant 7\big[m(2M+1) - 2\big(M + |B_m| + \min(B_m)\big) + 1\big] + M_{\mathcal{M}} - |B_b|,$$

where the term $M_{\mathcal{M}} - |B_b|$ is the number of clauses required to process the right hand side. The bounds stated in the lemma follow easily. □

We observe that Theorem 1 follows directly from this lemma.

Making use of specific problem-dependent structures, a number of redundant additional variables and clauses can usually be directly eliminated; we refer to [8].

BLPs with constant objective function can be transformed to CNF by applying the procedure described above to each of the inequalities separately. Note that it is important to keep track of negative coefficients: if a variable x_i has a negative coefficient, its associated proposition p_{x_i} must be negated in (9). BLPs with non-constant objective function can be transformed to CNF by adding a bound to the objective function and regarding it as a linear inequality. By performing a binary search on this bound and repeatedly solving the corresponding satisfiability problems an optimal solution can be obtained. It may be noted that the CNF formulas involved differ very little since only the subset of clauses corresponding to the artificial bound on the objective function changes.

We conclude this section by mentioning a modification of the transformation that restricts the number of additional clauses. Consider again (3). 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, this may be relaxed to $p_0^{(U)} \leftarrow (p_0^{(V)} \leftrightarrow \neg p_0^{(W)})$, as *implication* suffices. Similarly, we can replace the first equivalences by implications in expressions (5)–(8). Thus the number of

additional clauses is, roughly speaking, halved [8]. In the next section, we report on computational experience with both choices. Observe that the idea that is used here is similar to Wilson's modification [10] of the transformation by Blair et al. [1] to transform logical formulas to CNF.

4. Computational experience

Many combinatorial optimization problems can be expressed as BLPs that are *almost* equivalent to CNF formulas, 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 such problems with a logical solver. The aim of this section is to show that a logical solver can be more effective than a mathematical programming package for solving particular combinatorial optimization problems. In other cases however the transformation may require such a large number of auxiliary variables and clauses that the resulting satisfiability problem becomes intractable.

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, the objective is to assign each radio link a frequency, without violating any interference constraint and such that the number of distinct frequencies used is below a given maximum (provided by the user). An interference constraint is a triple (l, k, d_{lk}), where $d_{lk} \geqslant 0$ is the minimum distance required between the frequencies assigned to radio links l and k. Warners et al. [9] have developed various mathematical models for this problem. Here we use the following model. We introduce the proposition letters p_{lf} and q_f:

$$p_{lf} = \begin{cases} \textit{true} & \text{if frequency } f \text{ is assigned to} \\ & \quad \text{radio link } l, \\ \textit{false} & \text{otherwise,} \end{cases}$$

for all $l \in L$, $f \in F$

$$q_f = \begin{cases} \textit{true} & \text{if frequency } f \text{ is } \textit{not} \text{ assigned to} \\ & \quad \text{any radio link,} \\ \textit{false} & \text{otherwise,} \end{cases}$$

for all $f \in F$

Binary variables x_{lf} and z_f are associated with the propositions p_{lf} and q_f, respectively. Letting $F_{\min}$ be the minimal number of frequencies *not* to be used, the *FAP* is expressed as follows:

$$\bigvee_{f \in F} p_{lf}, \qquad \text{for all } l \in L, \tag{14}$$

$$\neg p_{lf} \vee \neg p_{kg}, \qquad \text{for all } (f, g) \text{ such that } |f - g| \leqslant d_{lk}, \tag{15}$$

$$\neg p_{lf} \vee \neg q_f, \qquad \text{for all } l \in L,\ f \in F, \tag{16}$$

with the additional constraint that *at least* $F_{\min}$ *propositions* q_f *must be true.* Here, (14) expresses that to each radio link a frequency must be assigned and (15) model the interference constraints. The clauses (16) keep track of which frequencies are assigned to at least one link, while the last constraint puts a bound on the number of distinct frequencies used. Only this constraint is not in CNF. It can be expressed as

$$\sum_{f \in F} z_f \geqslant F_{\min}, \tag{17}$$

so it can be transformed to CNF by our procedure.

For the instances of the *FAP* we used (see also [9]), $|F|$ is typically equal to 24. The number of additional variables and clauses to transform (17) are equal to 83 and 416 when using the transformation with equivalence; when restricting to implication the number of additional clauses reduces to 192. 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, (17) does not need to be transformed, while all clauses (14)–(16) can readily be written as linear inequalities. In Table 1 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 (14)–(16)); 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 those introduced during the transformation.

Table 1
Computational results on 4 *FAP*s with m variables and n constraints

G10.6	$m = 158, n = 1517$	HeerHugo	HeerHugo	CPLEX	CPLEX
$F_{\min}$	SAT ?	$\rightarrow$	$\leftrightarrow$	feas.	+ obj.
19	SAT	1	14	0.3	
20	SAT	13	10	0.4	
21	SAT	1	10	0.6	34
22	UNSAT	73	75	24	
23	UNSAT	2	2	2	
G16.6	$m = 232, n = 3225$	HeerHugo	HeerHugo	CPLEX	CPLEX
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	
G20.10	$m = 296, n = 7500$	HeerHugo	HeerHugo	CPLEX	CPLEX
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	
G20.6	$m = 411, n = 8979$	HeerHugo	HeerHugo	CPLEX	CPLEX
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	

Based on these experiments, we observe the following:

- For some of the larger *feasibility* 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.
- For the two largest problems, HeerHugo manages to solve *all five* feasibility problems in less time than CPLEX solves the optimization version of the problem. Thus HeerHugo is also able to solve the optimization version more efficient than CPLEX by applying a search over $F_{\min}$ values. For the two smaller problems this is not the case.
- The transformation to CNF with implication instead of equivalence, generally gives better results when the problem is satisfiable. This seems due to the fact that the transformation with implication allows more satisfiable assignments (if any).

From our experiments we draw the conclusion that a logical solver can be a useful complementary approach for solving specific binary programming problems. We observe that the results obtained by both CPLEX and HeerHugo could be substantially improved by using additional problem-specific information. For an overview of special-purpose algorithms for the *FAP*, see Tiourine et al. [7].

Acknowledgement

I would like to thank Jan Friso Groote for the initial idea that led to the main result, and Oliver Kullmann for his comments on an earlier version of the manuscript. The comments of an anonymous referee helped improving the presentation.

References

[1] C.E. Blair, R.G. Jeroslow, J.K. Lowe, Some results and experiments in programming techniques for propositional logic, Comput. Oper. Res. 13 (5) (1986) 633–645.

[2] S.A. Cook, The complexity of theorem proving procedures, in: Proc. 3rd Annual ACM Symposium on the Theory of Computing, New York, 1971, pp. 151–158.

[3] J.N. Hooker, Unpublished note.

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

[5] J.N. Hooker, Logic based methods for optimization, Technical Report, GSIA, Carnegie Mellon University, Pittsburgh, PA, 1994.

[6] J.N. Hooker, C. Fedjki, Branch-and-cut solution of inference problems in propositional logic, Ann. of Math. Artificial Intelligence 1 (1990) 123–139.

[7] S. Tiourine, C. Hurkens, 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, 1995.

[8] J.P. Warners, A linear-time transformation of linear inequalities into conjunctive normal form, Technical Report CS-R9631, CWI, Amsterdam, 1996.

[9] J.P. Warners, T. Terlaky, C. Roos, B. Jansen, A potential reduction approach to the frequency assignment problem, Discrete Appl. Math. 78 (1–3) (1997) 251–282.

[10] J.M. Wilson, Compact normal forms in propositional logic and integer programming formulations, Comput. Oper. Res. 17 (3) (1990) 309–314.