Computers Opns Res. Vol. 17, No. 3, pp. 309–314, 1990
Printed in Great Britain. All rights reserved

0305-0548/90 $3.00 + 0.00
Copyright © 1990 Pergamon Press plc

COMPACT NORMAL FORMS IN PROPOSITIONAL LOGIC AND INTEGER PROGRAMMING FORMULATIONS

J. M. WILSON*
Department of Management Studies, Loughborough University of Technology, Loughborough, Leicestershire LE11 3TU, England

(*Received March* 1988; *revised July* 1989)

Scope and Purpose—Various interconnections have been developed between artificial intelligence and integer programming which relate propositional logic theorem proving techniques to methods of solving integer programming problems. This paper will first describe a compact method of representing a logical proposition as a normal form. When this normal form is represented as a set of integer programming inequalities it is found that the set is very similar to that which would be produced by an operations research analyst formulating the original logical problem as an integer program. In addition, connections between propositional logic and the branch and bound method of integer programming are considered.

Abstract—An alternative linear time algorithm for achieving a clausal form equivalent to a given logical proposition is developed. The two algorithms are compared on a set of test problems and savings in computation are demonstrated. The style of the revised algorithm suggests a natural compact representation of logical propositions as integer programming inequalities and this representation is developed. Other elaborations of connections between logical propositions and mixed integer programming are developed involving conjunctive representations.

1. INTRODUCTION

Connections between mathematical logic and integer programming have been developed by various authors. Early work by Granot and Hammer [1] linked Boolean Logic and integer programming. Work by Jeroslow and Lowe [2, 3] continued developments and summaries of related work appear in Glover [4] and Williams [5]. In a recent paper Blair *et al.* [6] develop interconnections between propositional logic and integer programming and describe experimental results for solving satisfiability problems using integer programming solution techniques. This paper will describe a simplification possible in the method of representing the satisfiability problem in clausal form as an integer program. The simplification is motivated by consideration of operations research formulations of logical problems as integer programs and it groups clauses in an obvious way.

After a review of relevant propositional logic has been made, the algorithm of Blair *et al.* [6] for obtaining a conjunctive normal form of a proposition will be described and some improvements made to the algorithm. The connection between integer programming and propositional logic is explored further later in the paper where links to mixed integer programming are discussed.

2. AN ALGORITHM FOR OBTAINING A CONJUNCTIVE NORMAL FORM

Blair *et al.* [6] describe a linear time algorithm for producing a conjunctive normal form of a proposition. The *conjunctive normal form* of a proposition A is achieved when it is expressed as $A_1 \wedge \cdots \wedge A_n$ in which every subformula A_i $(1 \leqslant i \leqslant n)$ has the form

$$A_i = \bigvee_{j \in J_i} \pm P_j$$

where $J_i \subseteq \{1, \ldots, s\} \neq \phi$, where $\wedge$ and $\vee$ are the propositional connectives "and" and "or" respectively. In this form each P_j is a single letter (literal) and $\pm$ indicates that either P_j or $\neg P_j$ may occur, where $\neg$ is the symbol for "not". The reason for constructing a normal form of a proposition is that it is usually more straightforward to establish if the proposition is *satisfiable*

*J. M. Wilson received his BSc degree in mathematics from the University of Glasgow and his MSc and DPhil in operational research from the University of Sussex in 1972 and 1976. He has written on practical aspects of integer programming and various operational research topics. He is a Senior Lecturer in the Department of Management Studies at Loughborough University of Technology, England.

when it is presented as a normal form. A proposition is satisfiable if it is true for some assignment of the truth values "true" or "false" to its subformulas. It will later be established that a normal form has connections with the integer programming representation of a logical proposition and that connections continue in that satisfiability is linked to feasibility in integer programming.

The algorithm CNF1 of Blair *et al.* [6] for obtaining a conjunctive normal form, based on earlier work by Tseitin [7], proceeds by replacing each collection of clauses of a formula A separated by the same connective by a new clause consisting of a single letter and then replacing each collection of single letters connected by the same connective by a single letter proceeding iteratively from smaller to larger subformulas. The algorithm for a given formula A will now be described briefly.

Firstly, for each such subformula B, construct a list of clauses $C(B)$ and a literal B' which represents B, as follows:

(a) If B is a literal, $C(B)$ is empty and $B' = B$.
(b) If $B = C_1 \wedge \cdots \wedge C_n$ with all $C(C_i)$ and C_i' constructed $(1 \leqslant i \leqslant n)$, let B' be any new letter and then $C(B)$ is $\bigcup_{i=1}^n C(C_i)$ plus these clauses:

$$\neg C_1' \vee \neg C_2' \vee \cdots \vee \neg C_n' \vee B'$$
$$C_i' \vee \neg B' \quad (1 \leqslant i \leqslant n).$$

(c) If $B = C_1 \vee \cdots \vee C_n$ with all $C(C_i)$ and C_i' constructed $(1 \leqslant i \leqslant n)$, let B' be any new letter and then $C(B)$ is $\bigcup_{i=1}^n C(C_i)$ plus these clauses:

$$C_1' \vee C_2' \vee \cdots \vee C_n' \vee \neg B'$$
$$\neg C_i' \vee B' \quad (1 \leqslant i \leqslant n).$$

(d) If $B = \neg C$ then $C(B) = C(C)$ and $B' = \neg C'$.

The conjunctive normal form equivalent for A is the list of clauses $C(A)$ plus the one element clause A'.

Example

$$A = (P_1 \wedge P_2) \vee (P_3 \wedge P_4).$$

Here B_1 is $P_1 \wedge P_2$ and B_2 is $P_3 \wedge P_4$ and so $C(B_1)$ is $\neg P_1 \vee \neg P_2 \vee B_1'$, $P_1 \vee \neg B_1'$, $P_2 \vee \neg B_2'$ and $C(B_2)$ is $\neg P_3 \vee \neg P_4 \vee B_2'$, $P_3 \vee \neg B_2'$, $P_4 \vee \neg B_2'$, and $C(A)$ consists of $C(B_1)$, $C(B_2)$ plus the clauses

$$B_1' \vee B_2' \vee \neg A', \qquad \neg B_1' \vee A', \qquad \neg B_2' \vee A'.$$

Hence the conjunctive normal form for A consists of all clauses in $C(A)$ plus the unit clause A'.

Remark

Consideration of this example suggests that all that is required to represent $A = (P_1 \wedge P_2) \vee (P_3 \wedge P_4)$ is

$$P_1 \vee \neg B_1', \qquad P_2 \vee \neg B_1'$$
$$P_3 \vee \neg B_2', \qquad P_4 \vee \neg B_2',$$
$$\text{and} \quad B_1' \vee B_2'.$$

Here a set of clauses connected by disjunction can be treated as a single clause and this minor simplification turns out to be particularly helpful in computation.

The proposed algorithm CNF2 proceeds as follows. As in [6] for each subformula B of a given formula A a list of clauses $C(B)$ is constructed and a literal B' which represents B. However, clauses connected by disjunction are handled in a simpler way.

(a) If B is a literal $C(B)$ is empty and $B' = B$.
(b) If $B = C_1 \wedge \cdots \wedge C_n$ with all $C(C_i)$ and C_i' constructed $(1 \leqslant i \leqslant n)$ let B' be any new letter and then $C(B)$ is

$$\bigcup_{i=1}^{n} C(C_i)$$

plus these clauses: $C_i' \vee \neg B'$ $(1 \leqslant i \leqslant n)$.

(c1) If $B = C_1 \vee \cdots \vee C_n$ with all $C(C_i)$ and C_i' constructed $(1 \leqslant i \leqslant n)$ then $C(B)$ is

$$\bigcup_{i=1}^{n} C(C_i)$$

but no new literal is required and $C_1 \vee \cdots \vee C_n$ is treated as a single literal in the algorithm.

(c2) If $A = C_1 \vee \cdots \vee C_n$ with all $C(C_i)$ and C_i' constructed $(1 \leqslant i \leqslant n)$ and then $C(A)$ is $\bigcup_{i=1}^{n} C(C_i)$ plus the clause $C_1 \vee \cdots \vee C_n$.

(d) If $B = \neg C$ then $C(B) = C(C)$ and $B' = \neg C'$.

The conjunctive normal form for A consists of all clauses in $C(A)$.

Example

$$A = \{P_1 \wedge (P_2 \vee \neg P_3)\} \vee (\neg P_1 \wedge \neg P_2).$$

The algorithm first generates $C(P_2 \vee \neg P_3)$ which means simply regard A as $\{P_1 \wedge (P_2 \vee \neg P_3)\} \vee (\neg P_1 \wedge \neg P_2)$ as before but consider $P_2 \vee \neg P_3$ to be a single literal. Next it generates $C(B_1)$ and $C(B_2)$ for $B_1 = P_1 \wedge (P_2 \vee \neg P_3)$ and $B_2 = \neg P_1 \wedge \neg P_2$.

This yields

$$P_1 \vee \neg B_2$$
$$P_2 \vee \neg P_3 \vee \neg B_1$$
$$\neg P_1 \vee \neg B_2$$
$$\neg P_2 \vee \neg B_2$$

and regards A as $B_1 \vee B_2$.

It therefore concludes by generating $C(A)$ namely

$$B_1 \vee B_2.$$

Thus the conjunctive normal form of A is

$$(P_1 \vee \neg B_1) \wedge (P_2 \vee \neg P_3 \vee \neg B_1) \wedge (\neg P_1 \vee \neg B_2) \wedge (\neg P_2 \vee \neg B_2) \wedge (B_1 \vee B_2).$$

As CNF1 is a linear time algorithm and CNF2 uses the same fundamental iterative approach as CNF1, but with some simplifications, then CNF2 will also be a linear time algorithm. However, in general the proposed CNF2 approach will result in fewer clauses and fewer literals compared to the approach in [6].

A comparison of CNF1 and CNF2 was run on a Prime 750 computer. The results are presented in Table 1. A series of problems was generated randomly where a set of m clauses was generated with a fixed number of literals per clause with each literal being sampled without replacement and with equal probability from the n available. The connective linking each literal together and each clause together were also chosen randomly. The results in Table 1 show that for the tested problems CNF2 required (on average) less than half the computation time of CNF1 to produce the normal form and that the number of clauses in the normal form (on average) is also less than half that for CNF1. CPU times exclude input and output.

Table 1. Comparison of CNF1 and CNF2

	CPU time (sec)		CPU	Final No. of clauses		Clauses	Original size		No. of
Problem	CNF1	CNF2	CNF1/CNF2	CNF1	CNF2	CNF1/CNF2	m	n	literals per clause
1	1.39	0.66	2.11	329	145	2.27	30	30	4
2	1.24	0.55	2.25	234	92	2.54	30	30	4
3	1.23	0.54	2.28	230	77	2.99	30	30	4
4	1.22	0.55	2.22	233	52	4.48	30	30	4
5	1.26	0.56	2.25	228	98	2.33	30	30	4
6	0.92	0.57	1.61	260	96	2.71	30	30	5
7	0.91	0.57	1.60	255	114	2.24	30	30	5
8	1.36	0.64	2.13	288	111	2.59	40	30	4
9	1.37	0.64	2.14	329	127	2.59	40	30	5
10	1.23	0.56	2.20	219	97	1.89	30	40	5
11	1.06	0.56	2.26	220	74	2.97	30	40	5
	mean		2.06	mean		2.49			

Table 2. Further comparison of CNF1 and CNF2

Problem set	m	Ratio of CPU times CNF1/CNF2 Min	Max	Mean
1	15	1.14	1.34	1.25
2	30	1.67	1.71	1.69
3	60	2.12	2.23	2.19
4	90	1.88	1.95	1.92
5	120	2.15	2.34	2.00
Overall				1.88

In order to test CNF2 more thoroughly, 60 problems with 5 literals per clause over the range $m = 15, 30, 60, 90, 120, 150$ with 10 problems in each set were now tested. The results are presented in Table 2.

The 60 problems in Table 2 were also generated randomly by the same method as those in Table 1. For the smaller sized problems ($m = 15, 30$) the time saving by using CNF2 is not so marked but for larger problems ($m = 60, 90, 120, 150$) the computational advantage is generally a factor of about 2.0. Basic overheads of the algorithm may explain the lower advantage factor for $m = 15$, 30. Consideration of the steps of the two algorithms shows that the computational effort required by CNF1 is proportional to mk operations, where k is the mean number of literals per clause, whereas for CNF2 the computational effort is proportional to $0.5mk + 0.5m$ operations, as it can be assumed that the connectives $\vee$ and $\wedge$ occur with equal probability. Hence, the computational advantage of about 2.0 can be anticipated from the relative proportions of expected operations.

3. INTEGER PROGRAMMING REPRESENTATION OF CONJUNCTIVE NORMAL FORM

Following [6], the clause $\bigvee_{j\in J} \pm P_j$ for $J \subseteq \{1, \ldots, s\} \neq \phi$ is represented in an integer program (IP) by the constraint

$$\sum_{j\in J} z(\pm P_j) \geqslant 1$$

where $z(P_1), \ldots, z(P_s) \in [0, 1]$ are (binary) variables which have been introduced and $z(-P_j)$ is an abbreviation for $1 - z(P_j)$. A disjunction of clauses is represented by a set of constraints.

A proposition is satisfiable if and only if its IP representation has a feasible solution using the valuation '1' to correspond to 'T', and '0' to 'F'.

Blair *et al.* [6] gives the following canonical set of constraints to represent the conjunctive normal form of

$$A = \left(\bigwedge_{i\in I} B_i\right) \vee \left(\bigwedge_{j\in J} C_j\right)$$

with B_i, C_j clauses in $P_1, \ldots, P_s$ ($I \neq \phi$, $J \neq \phi$)

$$z\left(\bigwedge_{i\in I} B_i\right) + z\left(\bigwedge_{j\in J} C_j\right) + 1 - z(A) \geqslant 1 \tag{1}$$

$$\sum_{i\in I} (1 - z(B_i)) + z\left(\bigwedge_{i\in I} B_i\right) \geqslant 1 \tag{2}$$

$$z(B_i) + \left(1 - z\left(\bigwedge_{i\in I} B_i\right)\right) \geqslant 1 \quad (i \in I) \tag{3}$$

$$1 - z\left(\bigwedge_{i\in I} B_i\right) + z(A) \geqslant 1 \tag{4}$$

and three more constraints replacing B by C, I by J, and i by j in (2), (3), and (4).

Note. Throughout the IP representation it is assumed that constraints of the form $0 \leqslant z \leqslant 1$ are implicitly included for each [0, 1] variable.

Considering the IP representation from the operations research point of view, a typical OR problem might be to model the statement

"if project 1 or 2 is commenced then either project 3 is not commenced or both projects 4 and 5 are commenced".

A fairly natural way to write such a requirement is

$$P_1 \text{ or } P_2 \text{ implies (not } P_3) \text{ or } (P_4 \text{ and } P_5)$$

which is

$$A \equiv (\neg P_1 \wedge \neg P_2) \vee \neg P_3 \vee (P_4 \wedge P_5) \tag{5}$$

where the P's are used to denote the commencement of appropriate projects.

A typical operations research representation of the above would be

$$\delta_1 + \delta_2 + \bar{x}_3 \geqslant 1 \tag{6}$$

$$\bar{\delta}_1 + \bar{x}_1 \geqslant 1 \tag{7}$$

$$\bar{\delta}_1 + \bar{x}_2 \geqslant 1 \tag{8}$$

$$\bar{\delta}_2 + x_4 \geqslant 1 \tag{9}$$

$$\bar{\delta}_2 + x_5 \geqslant 1 \tag{10}$$

where variables δ_1 and $\delta_2 \in [0, 1]$ are the common IP notation variables used to denote possibilities, albeit in a slightly oblique way here, and $x_1, x_2, x_3, x_4, x_5 \in [1, 0]$ are used to indicate whether the five projects are undertaken ($\bar{\delta}_1 = 1 - \delta_1$).

Use of the algorithm CNF2 produces a normal form for the above problem which has the IP representation (11)–(15). Such a representation more closely resembles (6)–(10).

$$\bar{z}_1 + \bar{b}_1 \geqslant 1 \tag{11}$$

$$\bar{z}_2 + \bar{b}_1 \geqslant 1 \tag{12}$$

$$z_4 + \bar{b}_2 \geqslant 1 \tag{13}$$

$$z_5 + \bar{b}_2 \geqslant 1 \tag{14}$$

$$\bar{z}_3 + b_1 + b_2 \geqslant 1. \tag{15}$$

Hence the use of the algorithm CNF2 would seem to be desirable. For such IP representations it will still remain true that the linear relaxation of the IP form can be feasible when the list of clauses is not satisfiable (and the IP form is infeasible).

4. MIXED INTEGER PROGRAMMING

The modelling of the operations research problem discussed in Section 4

"if project 1 or 2 is commenced then either project 3 is not done or both projects 4 and 5 are commenced".

will involve two stages. The logic controlling the project solution has already been discussed but in addition usually there will be a series of consequences and conditions relating to the commencement or not of the various projects. Such conditions will lead to linear constraints in continuous variables, e.g. raw material constraints. The combination of IP constraints to handle the logic and continuous constraints to handle the other features gives rise to a mixed integer programming problem. This can be handled by the branch and bound technique (see for instance Williams [8]). However, the propositional normal form could be used within a branch and bound framework to control the search for solutions. The problem could then be broken up into three logical groupings. Each grouping would be backed up by sets of continuous constraints. The problem can be thought of as one collection of constraints but could be solved in a decomposed fashion with constraints being considered only within their sets, controlled by the logical variables. Overall control of optimality would be maintained by an objective function. Thus instead of the usual pair of subproblems created by branch and bound in a tree structure, the structure would be more complex. Figure 1 illustrates such a structure.

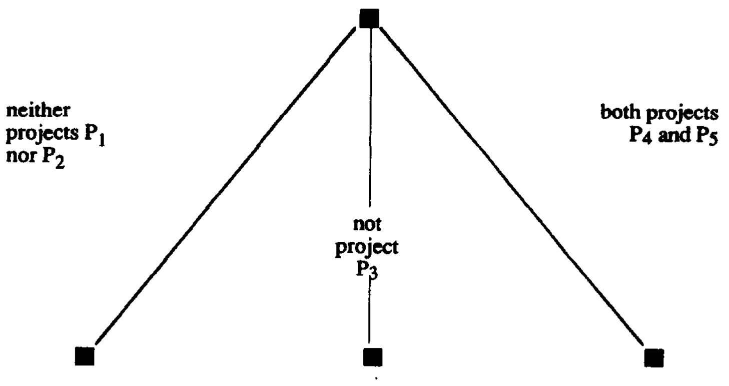
neither projects P_1 nor P_2
not project P_3
both projects P_4 and P_5

Fig. 1. A tree structure.

Other authors have investigated alternative branch and bound schemes and recent work by Beaumont [9] indicates that there may be merit in solving problems by decomposing them into disjunctive components. However, such work is experimental but the area of logically derived structures for branch and bound may be worthy of further investigation.

5. SUMMARY

A modified approach to obtaining a conjunctive normal form of a proposition has been proposed. This approach results in fewer clauses and literals than that of Blair *et al.* [6]. The representation of this normal form in integer programming inequalities results in a format which is "natural" and resembles the operations research formulation of the same problem. The computational results of Blair *et al.* [6] suggest that an IP approach based on LP is successful in testing for satisfiability large sets of clauses already in conjunctive normal form and this has implications for tests of satisfiability of Prolog language statements.

Considerable interest has developed in Horn Clauses because of the work of AI (e.g. Kowalski [10] in the development of the Prolog computing language [11] which represents its logical statements as Horn Clauses). Such clauses have a convenient representation in IP format and such representations are analogous with the Boolean approach of Ref. [1]. The IP approach to analysing Horn Clauses and the branch and bound approach for establishing their satisfiability may be surprisingly efficient as an alternative to the Prolog [11] approach because of evidence established by Blair *et al.* [6] for analysing propositional logic problems.

This paper has shown that the conjunctive normal form can be obtained more easily and the IP representation generated in a more compact way. Hence improvements should be achievable in the computational effort required to test for satisfiability sets of clauses whether in conjunctive normal form or not.

REFERENCES

1. F. Granot and P. L. Hammer, On the use of boolean functions in 0–1 programming. *Meth. Opns Res.* **12**, 157–184 (1972).
2. R. G. Jeroslow and J. K. Lowe, Experimental results on the new techniques for integer programming formulations. *J. Opl Res. Soc.* **36**, 393–403 (1985).
3. R. G. Jeroslow and J. K. Lowe, Modelling with integer variables. *Math. Program. Study* **22**, 167–184 (1984).
4. F. Glover, Future paths for integer programming and links to artificial intelligence. *Computers Opns Res.* **13**, 533–549 (1986).
5. H. P. Williams, Linear and integer programming applied to the propositional calculus. *Int. J. Syst. Res. Inform. Sci.* **2**, 81–100 (1987).
6. C. E. Blair, R. G. Jeroslow and J. K. Lowe, Some results and experiments in programming techniques for propositional logic. *Computers Opns Res.* **13**, 633–645 (1986).
7. G. S. Tseitin, On the complexity of derivation in propositional calculus. In *Automatization of Reasoning* (Edited by J. Siekman and G. Wrightson). Springer, Berlin (1983).
8. H. P. Williams, *Model Building in Mathematical Programming*. Wiley, London (1978).
9. N. Beaumont, A logical branch and bound algorithm. Paper presented at *Australian Society for Operations Research Conference on Mathematical Programming*, Melbourne, Australia, Oct. (1986).
10. R. Kowalski, *Logic for Problem Solving*. North-Holland, Amsterdam (1979).
11. *Turbo Prolog Owners Handbook*. Borland, Calif. (1984).