STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **2**, December 2014, pp 274–279.
Published online in International Academic Press (www.IAPress.org)

Efficient Binary Linear Programming Formulations for Boolean Functions

Frank Gurski [1, *]

[1] *Institute of Computer Science, Heinrich-Heine University of Düsseldorf, Germany*

Received: 25 May 2014; Accepted: 3 September 2014
Editor: Houduo Qi

Abstract A very useful tool when designing linear programs for optimization problems is the formulation of logical operations by linear programming constraints. We give efficient linear programming formulation of important n-ary boolean functions $f(x_1,\dots,x_n) = x_{n+1}$ such as conjunction, disjunction, equivalence, and implication using $n+1$ boolean variables $x_1,\dots,x_{n+1}$. For the case that the value $f(x_1,\dots,x_n)$ is not needed for further computations, we even give a more compact formulation. Our formulations show that every binary boolean function $f(x_1,x_2) = x_3$ can be realized by the only three boolean variables x_1,x_2,x_3 and at most four linear programming constraints.

Keywords Binary linear programming; Boolean functions

AMS 2010 subject classifications 90C05,06E30

DOI: 10.19139/soic.v2i4.83

1. Introduction

Linear programming is a powerful tool, studied for over 50 years, that can be used to define a lot of very important optimization problems [3, 6]. In a *linear programming problem (LP)* we are given a linear function $f:\mathbb{R}^n \mapsto \mathbb{R}$, $f(x_1,\dots,x_n) = c_1x_1+\ldots+c_nx_n = \sum_{i=1}^{n} c_ix_i$. Function f is denoted as *objective function* of the LP. Additionally, a set of constraints is given. In general a constraint either is an equality or an inequality which contains a

*Correspondence to: Heinrich-Heine University of Düsseldorf, Institute of Computer Science, Algorithmics for Hard Problems Group, D-40225 Düsseldorf, Germany. E-mail: `frank.gurski@hhu.de`

ISSN 2310-5070 (online) ISSN 2311-004X (print)

linear combination of the variables of f, i.e. the i-th constraint is of the form $a_{i,1}x_1 + \ldots + a_{i,n}x_n \geq b_i$. Some of these constraints may be very simple, since they require that some of the variables are not negative, e.g., $x_j \geq 0$. Constraints of the first type are denoted as *functional constraints* and the latter ones are denoted as *nonnegativity constraints*.

A proposal $x'_1, \ldots, x'_n$ of values for the n variables of f is called a *solution* and the number $f(x'_1, \ldots, x'_n)$ its *objective (function) value*. A solution is denoted as *feasible* if it satisfies all constraints. A feasible solution x is denoted as an *optimal solution* if the objective function value for x is the smallest (minimization problem) or the largest value (maximization problem) over the objective function values for all feasible solutions. The linear programming problem is, given an objective function and a finite set of constraints, to find an optimal solution. Using matrices, a linear program can be expressed as a task of minimizing $c^T x$ subject to the constraints $Ax \geq b$ and $x \geq 0$. This allows us to define the *size* of a linear program by $\text{size}(A) + \text{size}(b) + \text{size}(c)$. The size of an integer (by multiplying the rational coefficients adequate we can assume that we have given integer coefficients) is the size of its binary representation. The size of a vector or of a matrix is the sum of the sizes of its elements.

In *integer linear programming problems (IP)*, the variables are all required to be integers and in *binary linear programming problems (BIP)*, each variable can only take the two values, zero or one. While general linear programming can be solved in polynomial time by interior point methods or the ellipsoid method [3], solving integer programming and even binary integer programming is NP-hard [5].

The task to find equivalent linear programming formulations for optimization problems is often challenging, see e.g. [7], [8], and [2]. In this connection it is often useful to have formulation of logical operations by linear programming constraints. Examples are linear programming formulations for graph parameters in [7], an ILP approach for register allocation in [1], and the allocation of logical constraints in LP-solvers as e.g. CPLEX.

In this paper we give efficient binary linear programming formulations (BIP formulations) for several important boolean functions such as n-ary conjunction, n-ary disjunction, equivalence, and implication. Our results also imply that every binary boolean function $f(x_1, x_2) = x_3$ can be realized by the three variables x_1,x_2,x_3 and at most four conditions, which was unknown until now.

2. Realizing logic operations in linear programming

We extend the definition of the binary boolean functions **and** and **nor** given in [7] and of the functions **and**, **or**, and **not** given in [2] and [4] to all possible 16 binary boolean functions (cf. Table I) and their generalizations to n-ary boolean functions. Let x_i be boolean variables.

Table I. All 16 binary boolean functions $f(x_1, x_2)$.

x_1	x_2	0	$x_1 \wedge x_2$	$x_1 \wedge \neg x_2$	x_1	$\neg x_1 \wedge x_2$	x_2	$x_1 \oplus x_2$	$x_1 \vee x_2$	$\neg(x_1 \vee x_2)$	$x_1 \Leftrightarrow x_2$	$1 - x_2$	$x_2 \Rightarrow x_1$	$1 - x_1$	$x_1 \Rightarrow x_2$	$\neg(x_1 \wedge x_2)$	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

2.1. Negation

A **negation** $x_2 = \neg x_1$ can be realized by the following constraints.

$$\begin{array}{rcl} x_2 & = & 1 - x_1 \\ x_1, x_2 & \in & \{0, 1\} \end{array}$$

Note that for boolean variables x_i, we can always substitute x_i by $1 - x_i$ within our conditions to realize the negation of x_i without using the given additional constraint.

2.2. Conjunction

An **n-ary conjunction (and)** $x_{n+1} = \bigwedge_{i=1}^{n} x_i = x_1 \wedge x_2 \wedge \ldots \wedge x_n$ can be realized by $n + 1$ constraints as follows.

$$\begin{array}{rcl} x_{n+1} & \leq & x_1 \\ x_{n+1} & \leq & x_2 \\ \vdots & \vdots & \vdots \\ x_{n+1} & \leq & x_n \\ x_1 + x_2 + \ldots + x_n & \leq & x_{n+1} + n - 1 \\ x_1, x_2, \ldots, x_{n+1} & \in & \{0, 1\} \end{array}$$

The special case that we do not need the result of the n-ary conjunction for further computations in some variable and we only want to ensure that $(x_1 \wedge x_2 \wedge \ldots \wedge x_n)$ is satisfied, can be realized by the single constraint $\sum_{i=1}^{n} x_i \geq n$.

Although our first two given formulations imply that every binary boolean function can be defined by linear programming constraints, we suggest the following conditions in order to get more efficient formulations and to prove Theorem 1.

2.3. *Disjunction*

An n**-ary disjunction (or)** $x_{n+1} = \bigvee_{i=1}^{n} x_i = x_1 \vee x_2 \vee \ldots \vee x_n$ can be realized by $n+1$ constraints as follows.

$$\begin{array}{rcl} x_1 & \leq & x_{n+1} \\ x_2 & \leq & x_{n+1} \\ \vdots & \vdots & \vdots \\ x_n & \leq & x_{n+1} \\ x_1 + x_2 + \ldots + x_n & \geq & x_{n+1} \\ x_1, x_2, \ldots, x_{n+1} & \in & \{0,1\} \end{array}$$

The special case that we do not need the result of the n-ary disjunction for further computations in some variable and we only want to ensure that $(x_1 \vee x_2 \vee \ldots \vee x_n)$ is satisfied, can be realized by the single constraint $\sum_{i=1}^{n} x_i \geq 1$.

2.4. *Nand*

An n**-ary nand** $x_{n+1} = \neg(x_1 \wedge x_2 \wedge \ldots \wedge x_n)$ can be realized by $n+1$ constraints by substituting x_{n+1} by $1 - x_{n+1}$ in our solution for a n-ary conjunction.

The special case that we do not need the result of the n-ary nand for further computations in some variable and we only want to ensure that $\neg(x_1 \wedge x_2 \wedge \ldots \wedge x_n)$ is satisfied, can be realized by the single constraint $\sum_{i=1}^{n} x_i \leq n-1$.

2.5. *Nor*

An n**-ary nor** $x_{n+1} = \neg(x_1 \vee x_2 \vee \ldots \vee x_n)$ can be realized by $n+1$ constraints by substituting x_{n+1} by $1 - x_{n+1}$ in our solution for a n-ary disjunction.

The special case that we do not need the result of the n-ary nor for further computations in some variable and we only want to ensure that $\neg(x_1 \vee x_2 \vee \ldots \vee x_n)$ is satisfied, can be realized by the single constraint $\sum_{i=1}^{n} x_i \leq 0$.

2.6. *Exclusive Disjunction*

An **exclusive disjunction** $x_3 = x_1 \oplus x_2$ can also be defined by $x_3 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ and also by $x_3 = \neg(x_1 \Leftrightarrow x_2)$ or $x_3 = (x_1 + x_2) \bmod 2$ and be realized by four constraints.

$$\begin{array}{rcl} x_1 + x_2 + x_3 & \leq & 2 \\ -x_1 - x_2 + x_3 & \leq & 0 \\ x_1 - x_2 + x_3 & \geq & 0 \\ -x_1 + x_2 + x_3 & \geq & 0 \\ x_1, x_2, x_3 & \in & \{0,1\} \end{array}$$

The special case that we do not need the result of the binary exclusive disjunction for further computations in some variable and we only want to ensure that $(x_1 \oplus x_2)$ is satisfied, can be realized by the constraint $x_1 + x_2 = 1$.

2.7. Implication

An **implication** $x_3 = (x_1 \Rightarrow x_2)$ can also be defined by $x_3 = (\neg x_1 \vee x_2)$ and thus be defined by the shown conditions for a binary disjunction and substituting x_1 by $1 - x_1$ to realize $\neg x_1$.

$$\begin{array}{rcl} 1 - x_1 & \leq & x_3 \\ x_2 & \leq & x_3 \\ 1 - x_1 + x_2 & \geq & x_3 \\ x_1, x_2, x_3 & \in & \{0,1\} \end{array}$$

The special case that we do not need the result of the implication for further computations in some variable and we only want to ensure that $(x_1 \Rightarrow x_2)$ is satisfied, can be realized by the single constraint $x_1 \leq x_2$.

2.8. Equivalence

An **equivalence** $x_3 = (x_1 \Leftrightarrow x_2)$ can also be defined by $x_3 = (x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$ or $x_3 = \neg(x_1 \oplus x_2)$ and thus be realized by substituting x_3 by $1 - x_3$ in our solution for a binary exclusive disjunction.

$$\begin{array}{rcl} x_1 + x_2 - x_3 & \leq & 1 \\ x_1 + x_2 + x_3 & \geq & 1 \\ -x_1 + x_2 + x_3 & \leq & 1 \\ x_1 - x_2 + x_3 & \leq & 1 \\ x_1, x_2, x_3 & \in & \{0,1\} \end{array}$$

The special case that we do not need the result of the equivalence for further computations in some variable and we only want to ensure that $(x_1 \Leftrightarrow x_2)$ is satisfied, can be realized by the constraint $x_1 = x_2$.

2.9. Binary Boolean Functions

Our given formulations for logical operations imply for $n = 2$ that all possible 16 binary boolean functions $f : \{0, 1\}^2 \to \{0, 1\}$ which are summarized in Table 1 can be expressed very efficiently. The formulation of binary exclusive disjunction, or equivalently, the equivalence by only four conditions seems to be the hardest task within this result.

Theorem 1
Every binary boolean function $f : \{0, 1\} \times \{0, 1\} \to \{0, 1\}$, $f(x_1, x_2) = x_3$ can be realized by three variables x_1,x_2,x_3 and at most four conditions.

3. Conclusions and Outlook

We have given efficient linear programming formulation of several important n-ary boolean functions. Our results extend the definition of the binary boolean functions **and** and **nor** given in [7] and of the functions **and**, **or**, and **not** given in [2] and [4] and allow us to show that every binary boolean function $f(x_1, x_2) = x_3$ can be realized by the only three boolean variables x_1, x_2, x_3 and at most four linear programming constraints. We suppose that for $n = 2$ our solutions are best possible with respect to their size. For $n > 2$ the problem gets much more extensive.

REFERENCES

1. A.W. Appel and L. George. Optimal Spilling for CISC Machines with Few Registers. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, PLDI '01, pages 243–253. ACM, 2001.
2. D.-S. Chen, R.G. Batson, and Y. Dang. *Applied Integer Programming: Modeling and Solution*. Wiley, New York, 2010.
3. V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.
4. FICO$^{\text{TM}}$. MIP formulations and linearizations - Quick reference. Technical report, Fair Isaac Corporation, Warwickshire CV32 5YN, UK, 2009.
5. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.
6. M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey, editors. *50 Years of Integer Programming 1958-2008*. Springer-Verlag, 2010.
7. J. Luttamaguzi, M. Pelsmajer, Z. Shen, and B. Yang. Integer programming methods for several optimization problems in graph theory. In *Proceedings of the International Conference on Computers and Their Applications, CATA 2005*, pages 50–55. ISCA, 2005.
8. R. Vanderbei. *Linear Programming*. Springer-Verlag, Berlin, 2008.