stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE BW 10/71 MAY

JAC.M. ANTHONISSE

A NOTE ON PROHIBITIVE ALGORITHMS

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

1. Introduction.

The present note is based upon a number of observations by Edmonds [2]. There is an obvious finite algorithm to solve linear programming problems in zero-one variables. The cost of performing this algorithm, however, increases exponentially with the number of variables in the problem. So its application is restricted to problems of very limited size. It is by no means obvious whether or not there exists an algorithm whose cost increases only algebraically with the number of variables. The difference between algebraic and exponential order is often more crucial than the difference between finite and non-finite.

An algorithm prescribes elementary operations on the initial data of the problem and on intermediate results. Two cost-factors may be distinguished, the time to perform the elementary operations and the memory requirements to store initial data and intermediate results. It may be assumed that cost increases linearly with time and that time increases linearly with the number of elementary operations that are performed. Reference to a storage element may be considered as an elementary operation. As a storage element is not required unless referred to at least once, time and cost increase at least linearly with the storage requirements.

Throughout this note the term "prohibitive" will be used for an algorithm if the least upper bound on the cost of performing the algorithm increases (at least) exponentially with the number of variables in the problem. Thus an algorithm is "non-prohibitive" if the cost of solving any problem in n variables increases algebraically with n.

2. Boolean Functions.

A bivalent function in bivalent variables is defined to be a Boolean function, cf. Hammer and Rudeanu [3]. It may be assumed, without loss of generality, that each variable has the set {0,1} as its domain and that the same set constitutes the range of the function. With this convention, the domain of a Boolean function in n variables consists of the 2^n vectors

$$(x_1,\ldots,x_n)$$

with

$$x_j \in \{0,1\} \qquad (j = 1,\ldots,n).$$

Each vector corresponds to a vertex of the n-dimensional unit-cube. A Boolean function assigns a value from {0,1} to each of the 2^n vertices, consequently there are 2^{2^n} Boolean functions in n variables.

The straightforward way of storing a Boolean function is to list its value in the 2^n points of its domain. In this case memory requirements increase exponentially with n, thus all algorithms representing Boolean functions in the straightforward way are prohibitive.

Intuitively it is 'clear' that a sequence of 2^n bits to represent 2^n bivalent elements is very compact and not easily improved.

Theorem 1.

There exists no non-prohibitive method which can represent each Boolean function.

Proof.

In the straightforward method each function corresponds to a configuration of 2^n bits, containing the function itself. If less bits are used not the function itself but a configuration identifying the function is stored.

Different functions correspond to different configurations, as it must be possible to determine the function itself from its identification.
Consequently, 2^{2^n} bit configurations are required. If each configuration occupies b bits then 2^b different configurations are possible. If each configuration occupies at most B bits the number of different configurations is $\sum_{b=1}^{B} 2^b$.

As

$$2^{2^n} > \sum_{b=1}^{2^n-1} 2^b$$

a configuration of $b = 2^n$ bits will be required for at least one of the functions. This completes the proof.

It should be noted that, in any method to represent Boolean functions, at least half the number of functions correspond to configurations of length $b \geq 2^n-1$.

In many cases only a subclass from the class of all Boolean functions is considered. The linear functions

$$a_1x_1 \cup a_2x_2 \cup \ldots \cup a_nx_n \qquad *)$$

can be represented by the configuration

$$(a_1,a_2,\ldots,a_n),$$

requiring n bits for each function.

Now consider a situation where no Boolean function can be excluded but where it is known that some functions occur more frequently than the other functions. This knowledge could be exploited in the design of the method to represent Boolean functions.

---

*) $0.0 = 0.1 = 1.0 = 0 \quad 1.1 = 1$
$1 \cup 1 = 1 \cup 0 = 0 \cup 1 = 1 \quad 0 \cup 0 = 0$
} by definition

Assume $\{f \equiv 0 \text{ or } f \equiv 1\}$ occurs with probability $1 - 2^{-n}$ and $\{f \not\equiv 0 \text{ and } f \not\equiv 1\}$ occurs with probability 2^{-n}. Then a simple scheme requires $3 - 2^{-n}$ bits at the average occurrence of a function. This clearly is non-prohibitive on the average.

If a function has a positive probability of occurring it will occur, with probability 1, in the long run. Consequently, any method should anticipate the occurrence of each function, unless it has been shown that certain functions have zero probability of occurring.

An algorithm might be based upon the assumption that certain classes of functions have zero probability of occurring. The algorithm is terminated if a function from such a class occurs.

A practical consequence of the above remarks is that each algorithm prescribing the storage or manipulation of Boolean functions should be accompanied by a characterization of the functions to be expected.

3. Zero-One Linear Programming.

Consider the class of problems

$$\text{maximize} \sum_{j=1}^{n} c_j x_j \qquad \text{(i)}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \quad (i = 1,\ldots,m_1) \ , \qquad \text{(ii)}$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad (i = m_1+1,\ldots,m) \ , \qquad \text{(iii)}$$

$$x_j \in \{0,1\} \ (j = 1,\ldots,n) \ , \qquad \text{(iv)}$$

The coefficients a_{ij} and b_i represent a Boolean function f, defined in the following way:

$$f(x_1,\ldots,x_n) = \begin{cases} 1 & \text{if } (x_1,\ldots,x_n) \text{ satisfies (ii) and (iii),} \\ 0 & \text{otherwise.} \end{cases}$$

Thus f is the characteristic function of the feasible solutions of the problem.

The converse is also true, each Boolean function admits a representation by linear constraints. The constraint

$$\sum_{j=1}^{n} (2\alpha_j - 1)x_j \leq -1 + \sum_{i=1}^{n} \alpha_i$$

excludes vertex $(\alpha_1,\ldots,\alpha_n)$, and no other vertices.

The one to many correspondence between Boolean functions and linear programming problems in zero-one variables leads to the conclusion that there exists no non-prohibitive method of representing the initial data of the programming problems. So there exists no non-prohibitive algorithm which can solve each linear programming problem in zero-one variables.

The non-existence of a non-prohibitive algorithm does not imply that algorithms which can solve the general problem are useless. Such general algorithms can be non-prohibitive for subclasses of problems. They might also be non-prohibitive on the average, but this depends on the applications they are used for.
Several algorithms for the general problem are available, cf. [1]. Little is known about the subclasses for which these algorithms are non-prohibitive or non-prohibitive on the average.

The main conclusion to be drawn from the above is that each subclass of problems should be solved by a specific algorithm, exploiting all peculiarities of that subclass. This remark has been made before, cf. Hu [4]. Now assume that the class of all zero-one linear programming problems has been partitioned into subclasses, where the number of variables in a problem assumes arbitrarily large values in each subclass. Then it is prohibitive to determine the subclass to which a given problem belongs, or at least one of the subclasses admits prohibitive algorithms only.

4. References.

1. M.L. Balinski, K. Spielberg,
Methods for integer programming: algebraic, combinatorial and enumerative, chapter 7 of:
J.S. Aronofsky (ed.), Progress in Operations Research Vol. III, Wiley 1969.

2. J. Edmonds,
Paths, Trees, and Flowers,
Canadian Jl. of Math. 17(1965) 449-467.

3. P.L. Hammer, S. Rudeanu,
Boolean Methods in Operations Research,
Springer, 1968.

4. T.C. Hu
The development of network flow and related areas in programming,
Invited Lecture, 7th International Mathematical Programming Symposium, The Hague, 1970.