Note

A FORTRAN Program for the Solution of Simultaneous Linear Boolean Inequalities by the Algorithm of Hammer and Rudeanu

Hammer and Rudeanu have presented an algorithm for finding the solutions of a system of simultaneous linear inequalities in Boolean variables [1]. Our interest in this algorithm was generated by the discovery that the phase problem of X-ray crystallography for centrosymmetric space groups can be formulated in precisely this manner. The derivation of this equivalence has been presented elsewhere, together with an example to demonstrate that a real crystal structure can be solved by this method [2]. Here we wish to describe a modification that increases the efficiency of this algorithm, and its implementation in a FORTRAN program.

The Hammer–Rudeanu algorithm can be summarized as follows [1]:

(1) Convert each inequality to a canonical form (viz., for the *i*th inequality):

$$c^i_{i_1}\tilde{x}_{i_1} + c^i_{i_2}\tilde{x}_{i_2} + \cdots + c^i_{i_m}\tilde{x}_{i_m} \geqslant d^i,$$

with $c^i_{i_1} \geqslant c^i_{i_2} \geqslant \cdots \geqslant c^i_{i_m} > 0$. This is accomplished by a substitution of new variables $\tilde{x}_{i_j}$ for the original set. Each of the new variables is equal to one of the original variables or to the complement of one. The substitutions are in general different for each inequality.

(2) Classify each inequality according to its implications about the solution. Under some circumstances an inequality may be redundant; under others, a solution may be possible only if one or more variables take certain values. The latter are termed determinate cases. Indeterminate cases correspond to only partial information: a solution is consistent with only a finite (but hopefully relatively small) number of possible assignments of values to variables. It can also occur that an inequality is impossible to satisfy for *any* values of the unknowns.

(3) If no inequality is impossible to satisfy, certain variables are assigned values, and the inequalities rewritten to form a system in a smaller number of unknowns. Variables that are fixed by inequalities in determinate cases are assigned the unique values. If indeterminate cases only are present, one of the possibilities generated is selected and explored; subsequently, the other possibilities are explored, generating a tree.

(4) When a node of this tree is reached at which a solution has been deter-

mined or at which no solution is possible, the algorithm backtracks to the last node still possessing unexplored possibilities.

Hammer and Rudeanu have proved that their algorithm will generate all solutions.

Our program follows the procedure almost exactly, except for the classification of certain inequalities containing only a single term on the left side, e.g. $c \cdot x \geqslant d > 0$ ($x = 0$ or 1), with $c > d$. This inequality is determinate, since x must equal 1. However, the published procedure, if taken literally, would explicitly check the case $x = 0$, leading immediately to an absurdity. By reclassifying monomials of this type as determinate the extra check is avoided.

In constructing the program, it was found that a great increase in efficiency would result from care in storage allocation. The procedure involves extensive deletions and reinsertions of terms as the tree structure is traversed, as well as frequent searches for the largest term still present in an inequality at some node. The coefficients of each inequality are, therefore, stored in a doubly linked list, facilitating deletion, reinsertion, and searching.

The following information specifies each node of the tree:

1. A set of variables that are fixed, and their values.

2. A set of inequalities $\mathbf{A} \cdot \mathbf{X} \geqslant d$ ($\mathbf{A}$ is a real matrix, $\mathbf{X}$ is a Boolean vector, and d is a real vector). $\mathbf{A}$ is a submatrix of the original matrix. Some of its rows may have been deleted entirely as redundant; certain terms of other rows may have been deleted, as the corresponding variables become fixed in some partial solution.

3. Classification of all the "live" inequalities.

4. Two decisions: is no solution possible consistent with the current partial solution? Has a solution been reached?

5. Was the node entered from above or below?

6. Flow information. If the node was entered from above, branching information must be specified (the order in which the tree is to be traversed). If the node was entered from below, branching tables were constructed before.

7. What level is this node at? This serves as a pointer to the records of fixed variables and deleted terms and inequalities needed for backtracking.

The following specifies what to do at any node:

1. If no solution is possible, go back up to last node.

2. If a solution has been found, print it and go back to last node.

3. Is any inequality determinate? If so, fix the associated variables. Alter and reclassify the inequalities in which they occur, and record this information. Set branching tables so that any return to this node will be passed back on to its predecessor. Go back to step 1.

4. No variables have been fixed. If the node was entered from above, choose an inequality to branch on and initialize branching tables. If entered from below, take the next branch, as specified in the branching tables. This is done by fixing some variables, altering and reclassifying the inequalities, and recording the changes, then returning to step 1.

The program is coded entirely in FORTRAN IV. It is slightly specialized for the IBM 360 series by the use of type statements including lengths (e.g., LOGICAL*1). The program uses a sorting routine of Rochkind [3].

Although the program is too long (ca. 2700 cards) for reproduction of a listing here, copies are available upon request from the author.

Acknowledgment

Work supported by National Institutes of Health, U. S. Public Health Service research grant GM-16539 and U. S. National Science Foundation research grants GB-7875 and GJ-35327. This work made use of computer facilities supported in part by National Science Foundation grants NSF-GJ-34 and NSF-GU-3157.

References

1. P. L. Hammer (Ivanescu) and S. Rudeanu, "Boolean Methods in Operations Research and Related Areas," pp. 65 ff, Springer-Verlag, New York, 1968.
2. A. M. Lesk, Acta Cryst. **A28** (1972), 55.
3. M. M. Rochkind, *IEEE Trans. Computers* **C-19** (1970), 270.

Received: August 7, 1972

A. M. Lesk*

Department of Biochemical Sciences
Princeton University
Princeton, New Jersey 08540

*Present address: Department of Chemistry, Fairleigh Dickinson University, 1000 River Road, Teaneck, New Jersey 07666.

Printed in Belgium