stichting
mathematisch
centrum

MC

AFDELING MATHEMATISCHE BESLISKUNDE (DEPARTMENT OF OPERATIONS RESEARCH)

BW 64/77

JANUARI

B.L. FOX, J.K. LENSTRA, A.H.G. RINNOOY KAN, L.E. SCHRAGE

BRANCHING FROM THE LARGEST UPPER BOUND: FOLKLORE AND FACTS

Prepublication

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).

AMS (MOS) subject classification scheme (1970): 90C10

BRANCHING FROM THE LARGEST UPPER BOUND: FOLKLORE AND FACTS

B.L. FOX
Université de Montréal, Canada

J.K. LENSTRA
Mathematisch Centrum, Amsterdam, The Netherlands

A.H.G. RINNOOY KAN
Graduate School of Management, Delft, The Netherlands

L.E. SCHRAGE
University of Chicago, U.S.A.

ABSTRACT

Branch-and-bound algorithms are widely used to solve combinatorial maximization problems. At each step of such an algorithm a search strategy selects an active subset of feasible solutions for examination. In this paper we discuss the formal properties and the practical value of search strategies based on branching from the largest upper bound (BLUB strategies). We investigate conditions under which BLUB strategies are optimal in the sense that they minimize the number of subsets generated. Counterexamples show that the conditions given in the literature are not strong enough and a correct optimality condition is formulated. Finally, we argue that the practical objections raised against BLUB strategies are not necessarily convincing.

KEY WORDS & PHRASES: *branch-and-bound algorithm, search strategy, mixed integer programming*

NOTE: This report is not for review; it has been submitted for publication in a journal.

1. INTRODUCTION

Let us consider branch-and-bound algorithms as applied to maximization problems. One of the components of such an algorithm is a search strategy, that at each step selects an active subset of feasible solutions for examination. The search strategies based on *branching from the largest upper bound* (BLUB strategies) are often credited with certain optimality properties. For instance, Lawler and Wood [6,p.712] comment on BLUB strategies:

> Suppose that for any given problem the set of new bounding problems is uniquely determined. Then this policy has the advantage that the total amount of computation is minimized, in the sense that any branching operation performed under this policy must also be performed under any other policy.

Garfinkel and Nemhauser [4,p.149] give the following exercise:

> Assume that all aspects of a branch and bound algorithm have been determined, except for the branching rule. One is interested in devising a branching rule that minimizes the number of vertices considered. If the original problem is to find all optimal solutions to
>
> $$\max z(x), \quad x \in S$$
>
> show that the appropriate rule is "branch to a vertex having the largest upper bound."

In Section 2, counterexamples show that neither of these characterizations is completely accurate and a correct optimality condition is formulated. In Section 3, we consider the practical value of BLUB strategies and argue against some traditional criticisms.

2. FORMAL PROPERTIES OF BLUB STRATEGIES

The essential components of a branch-and-bound algorithm for a maximization problem are the following.

- A *lower bound* LB on the value of the optimal solution is provided by the value of the best feasible solution found so far.
- A *branching rule* β ("separation rule" in [4]) associates a family $\beta(S)$ of subsets to a subset S of feasible solutions such that $\bigcup_{S' \in \beta(S)} S' = S$; the subsets S' are the *descendants* of the *parent* subset S.
- A *bounding rule* UB provides an *upper bound* UB(S) on the value of each solution in a subset S generated by β.
  - Computation of UB(S) is assumed to include feasibility tests (if any) and can therefore lead to *improvement* of LB.
  - *Elimination* of S occurs if $UB(S) \leq LB$.
- A *search strategy* σ ("branching rule" in [4]) selects a currently active subset for examination, where:
  - *active* subsets are generated subsets which have neither been eliminated nor led to branching;
  - *examination* of S is understood to imply both determination of $\beta(S)$ and computation of UB(S') for all $S' \in \beta(S)$.

The rules β and UB are assumed fixed, independent of σ. Two common types of search strategies are *jumptrack* strategies, whereby any active subset can be selected, and *backtrack* strategies, which, after examination of $S' \in \beta(S)$, examine all descendants $S'' \in \beta(S')$ before returning to the other members of $\beta(S)$. A jumptrack strategy is called a *BLUB strategy* if it prescribes the examination of an active subset with largest upper bound; a strategy is *optimal* if it minimizes the number of subsets generated to find an optimal solution (including verification of its optimality).

An actual branch-and-bound computation can be conveniently represented by means of a *search tree*. Initially, the tree consists of a single node corresponding to the set of all feasible solutions. The generation of a subset S is represented by the creation of a node corresponding to S and an edge between this node and its parent.

In this section we will investigate the optimality of BLUB strategies. In the first place, the presence of *ties among upper bounds* can lead to

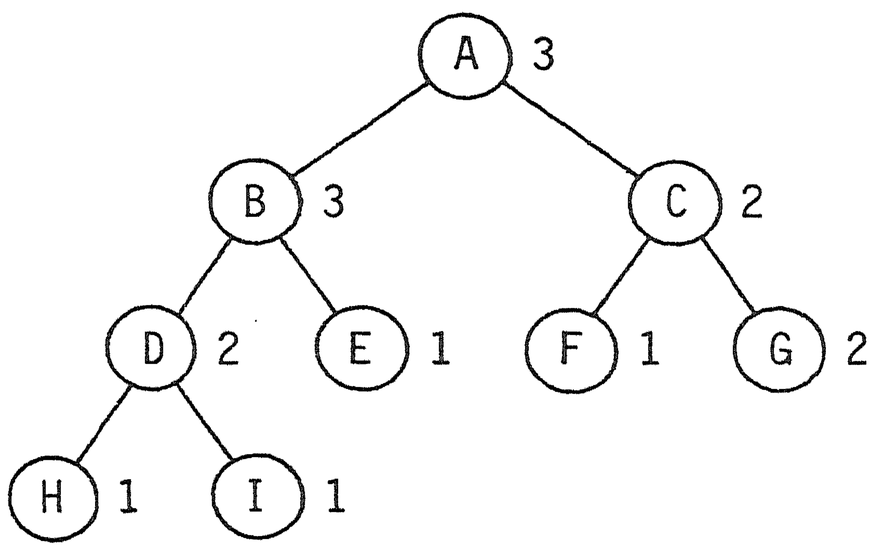

Figure 1 Search tree for the example.

situations in which *some* BLUB strategies are non-optimal, as demonstrated by the following example.

Example. Consider the search tree given in Figure 1. The nodes correspond to subsets denoted by capitals and are labelled by the values of their upper bounds. Let us assume that the unique optimal solution with value 2 is contained in G and is found during computation of UB(G). Consider the following search strategies:

- σ_1 examines A,B,C in that order and generates B,C,D,E,F,G;
- σ_2 examines A,B,D,C in that order and generates B,C,D,E,H,I,F,G;
- σ_3 examines A,C,B in that order and generates B,C,F,G,D,E.

Obviously, σ_1 is an optimal BLUB strategy, σ_2 is a non-optimal BLUB strategy, and σ_3 is an optimal non-BLUB strategy. □

In the second place, there are situations in which *all* BLUB strategies may be non-optimal. At each stage of the tree search, we define the *complete history* as all information obtained so far and the *direct history* as the information obtained along the path from the initial node to the node under examination. Up to now, we have tacitly assumed that branching and bounding rules only depend on the direct history. Let us quote two instances of actual branch-and-bound algorithms for which this assumption is false.

- Geoffrion's mixed integer programming algorithm [5] (see also [4, Section 4.7]) computes strong upper bounds via the appropriate linear programming relaxations at, say, every sixteenth node; at all other nodes only a derived surrogate constraint is used. In this case, UB depends on the complete history.

- The MPSX mixed integer programming algorithm [2] allows the use of a variation of "pseudo-costs". For example, the attractiveness of branching on a certain variable may be measured by the average reduction in upper bound over all previous nodes in which branching on that variable took place. Thus, β is allowed to depend on the complete history.

It is easily checked that branching and bounding rules dependent on the complete history may destroy optimality with respect to all BLUB strategies.

Example (continued). Suppose that at the third node generated a weak upper bound is computed instead of a strong one. Suppose further that the weak and strong bounds are the same except at node D, where the weak bound is 3 and the strong bound is 2, as above. The unique BLUB strategy is the non-optimal strategy σ_2 and the unique optimal strategy is the non-BLUB strategy σ_3. □

When *is* branching from the largest upper bound optimal? Consider the following conditions:

(1) no ties occur among upper bounds;

(2) branching and bounding rules depend only on the direct history.

If (1) and (2) hold, then all BLUB strategies are optimal. If (1) is violated but (2) holds, then we have that:

- at least one BLUB strategy is optimal;
- if a BLUB strategy generates n nodes, e of which have upper bounds equal to the optimal solution value and d of which are descendants of such nodes but have smaller upper bounds, then any search strategy has to generate at least n-(d+e-1) nodes, examining at least n-(e-1) of them.

The latter observation implies that, under condition (2), any BLUB strategy is not far from optimal, unless there are many tied bounds. The proofs of the above results are simple and can be left to the reader.

3. PRACTICAL VALUE OF BLUB STRATEGIES

If condition (2) holds, the above propositions suggest the use of BLUB strategies. Even if (2) does not hold, BLUB may be a good heuristic. The case for BLUB seems especially strong if we terminate when the difference between the current largest upper bound and LB is less than a given tolerance. Yet no commercial code that we know of uses BLUB.

Traditional objections raised against BLUB strategies are that they require:

(A) excessive time to find an active node with largest upper bound;

(B) excessive bookkeeping;

(C) excessive storage.

Objection (A) seems hardly serious. By the use of existing list processing algorithms, the family of currently active nodes can be kept ordered according to nonincreasing upper bounds, and deletion and insertion of nodes can be accomplished in time proportional to the logarithm of the family size [3;1].

In contrast to backtrack strategies, jumptrack strategies such as BLUB cannot be implemented in a straightforward recursive manner. This makes BLUB strategies relatively complex to program and adds to the bookkeeping overhead. We would grant that on small-scale or one-shot applications BLUB does not appear worth-while for this reason. Let us assume then that we are contemplating the design of a large-scale production code.

Probably, the most involved bookkeeping arises when it would be very inefficient to compute bounds from scratch at each node. For example, suppose that bounds are computed via linear programming. For each generated node, we either store the corresponding basis inverse directly or provide sufficient information so that it can be easily reconstructed. Inverses for adjacent nodes generally differ only little, so that relatively few (strategically placed) inverses need to be stored explicitly.

In such situations, objection (B) may be valid. Still, it would require actual computational experiments to investigate if the trade-off between tree size and bookkeeping overhead works out to the disadvantage of BLUB strategies.

Similar remarks apply to objection (C). In general, jumptracking will

require more storage than backtracking. This is a disadvantage only if both:

(a) a backtrack strategy permits everything to be done in core;

(b) accessing secondary storage takes a considerable amount of time (*i.e.*, buffering with negligible swap times is not feasible), or secondary storage space cannot effectively be regarded as infinite.

With respect to some large-scale applications, (a) does not hold and it seems that (b) has not been seriously investigated. Buffering may be facilitated and secondary storage requirements reduced if we alternate between jumptracking and backtracking.

To summarize: we do not find the objections raised against BLUB strategies necessarily convincing and we believe that they merit serious empirical investigation.

REFERENCES

1. A.V. AHO, J.E. HOPCROFT, J.D. ULLMAN (1974) *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass.

2. M. BÉNICHOU, J.M. GAUTHIER, P. GIRODET, G. HENTGÈS, G. RIBIÈRE, O. VINCENT (1971) Experiments in mixed-integer linear programming. *Math. Programming* 1,76-94.

3. B.L. FOX (1970) Accelerating list processing in discrete programming. *J. Assoc. Comput. Mach.* 17,383-384.

4. R.S. GARFINKEL, G.L. NEMHAUSER (1972) *Integer Programming*. Wiley, New York.

5. A.M. GEOFFRION (1969) An improved implicit enumeration approach for integer programming. *Operations Res.* 17,437-454.

6. E.L. LAWLER, D.E. WOOD (1966) Branch-and-bound methods: a survey. *Operations Res.* 14,699-719.