Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems

WALTER H. KOHLER

University of Massachusetts, Amherst, Massachusetts

AND

KENNETH STEIGLITZ

Princeton University, Princeton, New Jersey

ABSTRACT. Branch-and-bound implicit enumeration algorithms for permutation problems (discrete optimization problems where the set of feasible solutions is the permutation group S_n) are characterized in terms of a sextuple (B_p, S, E, D, L, U), where (1) B_p is the branching rule for permutation problems, (2) S is the next node selection rule, (3) E is the set of node elimination rules, (4) D is the node dominance function, (5) L is the node lower-bound cost function, and (6) U is an upper-bound solution cost. A general algorithm based on this characterization is presented and the dependence of the computational requirements on the choice of algorithm parameters S, E, D, L, and U is investigated theoretically. The results verify some intuitive notions but disprove others.

KEY WORDS AND PHRASES: discrete optimization, branch-and-bound implicit enumeration algorithms, permutation problems, sextuple characterization, computational requirements, theoretical comparison

CR CATEGORIES: 5.40, 5.49

1. *Introduction*

Branch-and-bound implicit enumeration algorithms have recently emerged as the principal general method for finding optimal solutions for discrete optimization problems. Application of the branch-and-bound technqiue has grown rapidly and a complete list of references would exceed several hundred. Representative examples of this thrust include: flow-shop and job-shop sequencing problems [8, 15, 16], traveling salesman problems [7], general quadratic assignment problems [13], and integer programming problems [5]. Branch-and-bound algorithms, while usually more efficient than complete enumeration, have computational requirements that frequently grow as an exponential or high degree polynomial with problem size n. In these cases, their usefulness is limited to small size problems (relative to the size of most practical problems). The identification and implementation of computationally efficient algorithms is essential. Although the branch-

This work was supported by the National Science Foundation under grant NSF-GJ-965, and the US Army Research Office–Durham under contract DAHC04-69-C-0012.

Authors' addresses: W. H. Kohler, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01002; K. Steiglitz, Department of Electrical Engineering, Princeton University, Princeton, NJ 08540.

and-bound method has been surveyed and generalized by numerous authors [1, 2, 10–12, 14], very little has been proved about the relative computational requirements as a function of the choice of algorithm parameters. One notable exception is the recent work of Fox and Schrage [4]. They compared theoretically the relative number of nodes examined by branch-and-bound integer programming algorithms for three different branching strategies (next node selection rules).

To effectively compare algorithms, it is first necessary to establish some general classification scheme. In this paper we propose a classification scheme for branch-and-bound algorithms based on a sextuple of parameters (B,S,E,D,L,U), where (1) B is the branching or partitioning rule, (2) S is the next node selection rule, (3) E is the set of node elimination rules, (4) D is the node dominance function, (5) L is the node lower-bound cost function, and (6) U is an upper-bound solution cost. The general framework for integer programming algorithms recently introduced by Geoffrion and Martsen [5] is suggestive of the above scheme, but is not as explicit. We will demonstrate how the sextuple (B,S,E,D,L,U) can be used to describe the class of common optimum producing branch-and-bound algorithms for permutation problems (where the set of feasible solutions is the permutation group S_n). (This framework can be easily extended to describe suboptimal producing branch-and-bound algorithms as well [9]). The class of permutation problems includes flow-shop sequencing and quadratic assignment problems as special cases. Within this basic framework, we investigate theoretically the relative number of generated nodes and active nodes as functions of the choice of sextuple parameters S, E, D, L, and U. The results of this analysis verify some of our intuitive notions and disprove others.

2. *General Characterization*

Although the following definitions may seem too restrictive and the notation quite heavy, we found it necessary in order to be formally precise and correct.

2.1. Preliminary Definitions and Notation

(1) A *permutation problem* of *size n* is a combinatorial optimization problem defined by the triple (S_n,X,f) with the following connotation:

(i) The *solution space* $S_n \triangleq \{\text{permutations of } n \text{ objects}\} = \{\pi\}$.

(ii) The *parameter space* X, each point $x \in X$ represents admissible "data" for the problem.

(iii) The *cost function* $f : S_n \times X \to R$, where $f(\pi, x)$ is the *cost* of solution π with parameter value x.

(2) A *globally optimal* solution for parameter value x is a solution $\pi^* \in S_n$ such that $f(\pi^*, x) \leq f(\pi, x) \forall \pi \in S_n$.

(3) Given a set $M = \{i_1, i_2, \cdots, i_m\} \subseteq N \triangleq \{1, 2, \cdots, n\}$, π^M denotes a permutation of the set M.

(i) If $M = N$, then π^M will be called a *complete permutation* on N.

(ii) If $M \subseteq N$, then π^M will be called a *partial permutation* on N.

(iii) If $M = \phi$, then $e \triangleq \pi^\phi$.

(4) Given $\pi_r^M \triangleq (r_1, r_2, \cdots, r_m)$.

(i) $|\pi_r^M| \triangleq$ size of partial permutation $\pi_r^M = |M| = m$.

(ii) $\bar{M} \triangleq N - M$.

(iii) If $l \in \bar{M}$, let $\pi_r^M \circ l \triangleq (r_1, r_2, \cdots, r_m, l)$.

(iv) If π_s^K is a permutation of the set $K \subseteq \bar{M}$ and $\pi_s^K = (s_1, s_2, \cdots, s_k)$, let $\pi_r^M \circ \pi_s^K \triangleq (\cdots((\pi_r \circ s_1) \circ s_2) \cdots \circ s_k)$.

(v) $\{\pi_r^M \circ\} \triangleq \{\pi_r^M \circ \pi_s^{\bar{M}} \mid \pi_s^{\bar{M}} \text{ is any permutation of } \bar{M}\}$
$\triangleq$ the *completion of* π_r^M
= the set of all complete permutations beginning with partial permutation π_r^M.

(5) Given a set Y of partial permutations on N,

$$\{Y \circ\} \triangleq \{\bigcup \{\pi_y^K \circ\} \mid \pi_y^K \in Y\}$$
$$\triangleq \text{the } \textit{completion} \text{ of } Y.$$

(6) Given π_r^M, $M \subseteq N$.

(i) A *descendant of* π_r^M is any partial permutation $\pi_s^P = \pi_r^M \circ \pi_t^K$ with $K \subseteq \bar{M}$.

(ii) An *immediate descendant* of π_r^M is any descendant $\pi_s^P = \pi_r^M \circ \pi_t^K$ such that K is a singleton set and $K \subseteq \bar{M}$, i.e. $\pi_s^P = \pi_r^M \circ l$ for some $l \in \bar{M}$.

(iii) An *ancestor* of π_r^M is any partial permutation π_q^L such that $\pi_r^M = \pi_q^L \circ \pi_t^K, K \neq \phi$.

(iv) (The *immediate ancestor* of π_r^M is the ancestor π_q^L such that K is the singleton set, i.e. if $\pi_r^M = (r_1, r_2, \cdots, r_m)$, then the immediate ancestor is $\pi_q^L = (r_1, r_2, \cdots, r_{m-1})$.

(7) A one-to-one onto mapping (bijection) is established between the set of partial solutions of a permutation problem and the set of partial permutations of N. Each partial solution is then denoted by its corresponding partial permutation π_r^M. The terms *partial solution* and *partial permutation* will be used synonymously.

2.2. Sextuple Characterization: (B_p,S,E,D,L,U). The optimum producing branch-and-bound algorithms commonly used to solve the permutation problem (S_n,X,f) [1, 10] can be characterized in terms of the sextuple (B_p,S,E,D,L,U). The sextuple parameters are defined as follows:

(1) *Branching rule* B_p defines the branching process for permutation problems. The object of branching is to partition the set of complete solutions S_n into disjoint subsets. These subsets are represented by nodes. Each node is labeled by a permutation $\pi_y^{M_y}$ defined on a set M_y for $M_y \subseteq N = \{1, 2, \cdots, n\}$. The node labeled $\pi_y^{M_y}$ represents the set of complete solutions $\{\pi_y^{M_y} \circ\}$. Partial solution $\pi_y^{M_y}$ is an ancestor of each solution in the set $\{\pi_y^{M_y} \circ\}$. Branching at node $\pi_b^{M_b} = (b_1, b_2, \cdots, b_k)$ is the process of partitioning set $\{\pi_b^{M_b} \circ\}$ into $\{\pi_b^{M_b} \circ\} = \bigcup_{l \in \bar{M}_b} \{(\pi_b^{M_b} \circ l) \circ\}$. $\pi_b^{M_b}$ is called the *branching node.* It follows that $\{(\pi_b^{M_b} \circ i) \circ\} \cap \{(\pi_b^{M_b} \circ j) \circ\} = \phi$ for $i, j \in \bar{M}_b$ and $i \neq j$. Nodes $\pi_b^{M_b} \circ l$, $l \in \bar{M}_b$, are the immediate descendants of branching node $\pi_b^{M_b}$. These nodes are said to be *generated* at *branching step* $\pi_b^{M_b}$. (To help simplify this cumbersome notation, π_y will be used as a shorthand notation for $\pi_y^{M_y}$, with the set M_y understood.)

(2) *Selection rule* S is used to choose the next branching node, π_b, from the set of currently active nodes. Node π_y is *currently active* during the execution of algorithm (B_p,S,E,D,L,U) if and only if it has been generated but not yet eliminated or branched-from. The immediate descendants of branching node π_b are generated in lexicographic order. Descendant $\pi_b \circ l$, $l \in \bar{M}_b$, is added to the set of currently active nodes if and only if it is not eliminated by one of the node elimination rules in E. The algorithm is always initiated by selecting $e = \pi^\phi$ as the first branching node. (e is also assumed to be the first node generated.)

Although many other variations are possible, the common selection rules are designated as follows:

(i) $S = LLB$ (*L*east *L*ower-*B*ound Rules). Select the currently active node π_a with the least lower-bound cost $L(\pi_a)$. In the case of ties, either select the node that was generated first, LLB_{FIFO}, or last, LLB_{LIFO}.

(ii) $S = FIFO$ (*F*irst-*I*n-*F*irst-*O*ut Rule). Select the currently active node that was generated first.

(iii) $S = LIFO$ (*L*ast-*I*n-*F*irst-*O*ut Rule). Select the currently active node that was generated last, but skip those active nodes that are complete solutions unless there are no active nodes that are incomplete.

The branch-and-bound algorithm is assumed to terminate if the next branching node is a complete solution. When termination of (B_p,S,E,D,L,U) occurs in this manner, it will be proved in Section 3 that at least one of the currently active nodes is an optimal complete solution.

(3) *Dominance function D* is a binary relation defined on the set of partial solutions of (S_n, x, f). Given partial solutions (nodes) π_y and π_z, let $\hat{\pi}_y^N$ and $\hat{\pi}_z^N$ be minimum cost complete solutions beginning with π_y and π_z respectively. That is, $\hat{\pi}_y^N \in \{\pi_y \circ\}$ and $\hat{\pi}_z^N \in \{\pi_z \circ\}$ with $f(\hat{\pi}_y^N, x) = \min\{f(\pi_w, x) \mid \pi_w \in \{\pi_y \circ\}\}$ and $f(\hat{\pi}_z^N, x) = \min\{f(\pi_w, x) \mid \pi_w \in \{\pi_z \circ\}\}$. $\mathfrak{D}$ is the transitive binary relation defined on the set of partial solutions of (S_n, x, f) such that $\pi_y \mathfrak{D} \pi_z$ if and only if $f(\hat{\pi}_y^N, x) \leq f(\hat{\pi}_z^N, x)$. D is a subset of $\mathfrak{D}$ chosen to have the following properties for all partial solutions π_w, π_y, and π_z:

(i) (Subset property). $\pi_y D \pi_z$ only if $M_y \supseteq M_z$.

(ii) (Transitivity property.) $\pi_w D \pi_y$ and $\pi_y D \pi_z$ only if $\pi_w D \pi_z$.

(iii) (Consistency property). $\pi_y D \pi_z$ only if $L(\pi_y) \leq L(\pi_z)$.

Since $D \subseteq \mathfrak{D}$, it follows immediately that $\pi_y D \pi_z$ only if the minimum cost complete solution descended from node π_y has cost less than or equal to the minimum cost complete solution descended from node π_z. When this is true, π_y is said to *dominate* π_z. It is usually assumed that D is defined to contain all pairs $\pi_y D \pi_z$ with $L(\pi_y) \leq L(\pi_z)$ and $M_y = N$. In general, D must be chosen such that a test for inclusion in D is computationally feasible.

(4) *Lower-bound function L* assigns to each partial solution π_y a real number $L(\pi_y)$ representing a lower-bound cost for all complete solutions in the set $\{\pi_y \circ\}$. L is required to have the following properties:

(i) If π_z is a descendant of π_y, then $L(\pi_z) \geq L(\pi_y)$.

(ii) For each complete solution π_y^N, $L(\pi_y^N) = f(\pi_y^N, x)$.

(5) *Upper-bound* cost U is the cost of the currently known best complete solution π_u^N. π_u^N is updated during execution of the algorithm whenever a complete solution with cost less than U is generated. If no complete solution is known, U is assumed to be some cost, $U \triangleq \infty$, greater than all possible costs.

(6) *Elimination rules E* are a set of rules for using dominance function D and upper-bound cost U to render inactive (*eliminate*) newly generated and currently active nodes. Given algorithm (B_p, S, E, D, L, U), let π_b be the current branching node, $\pi_b \circ l$ an immediate descendant of π_b, and π_a a member of the currently active set at branching step π_b, $\pi_a \neq \pi_b$. E represents some subset of the following four rules:

(i) *U/DBAS* (*u*pper-bound tested for dominance of *d*escendants of *b*ranching node and members of currently *a*ctive *s*et). If $L(\pi_b \circ l) > U$, then all members of $\{(\pi_b \circ l) \circ\}$ have costs greater than the upper-bound solution π_u^N, where $U \triangleq f(\pi_u^N, x)$. In this case $\pi_b \circ l$ is eliminated after being generated and before becoming active. If $L(\pi_a) > U$, then all members of $\{\pi_a \circ\}$ have costs greater than the upper-bound solution π_u^N. π_a is then removed (eliminated) from the active set.

(ii) *AS/DB* (*a*ctive node *s*et tested for dominance of *d*escendants of *b*ranching node). Each currently active node is tested for dominance of each descendant of the branching node. If $\pi_a D (\pi_b \circ l)$ then $\pi_b \circ l$ is eliminated after being generated and before becoming active.

(iii) *BFS/DB* (*b*ranched-*f*rom node *s*et tested for dominance of *d*escendants of *b*ranching node). Each previous branching node is tested for dominance of the descendants of the current branching node. If π_p is a previous branching node and $\pi_p D (\pi_b \circ l)$, then $\pi_b \circ l$ is eliminated after being generated and before becoming active.

(iv) *DB/AS* (*d*escendants of *b*ranching node tested for dominance of currently *a*ctive node *s*et). Each descendant of the current branching node is tested for dominance of each currently active node. If $(\pi_b \circ l) D \pi_a$, then π_a is removed (eliminated) from the set of currently active nodes.

If E contains rules *AS/DB* and *DB/AS*, then the set of active nodes may depend on the order in which *AS/DB* and *DB/AS* are applied. We will assume that *AS/DB* is applied before *DB/AS*.

2.3. DESCRIPTIVE NOTATION AND CHART REPRESENTATION. The following notation will be used to describe the detailed operation of algorithm $BB \triangleq (B_p,S,E,D,L,U)$:

$BFS(\pi_b)$ $\triangleq$ set of previous branching nodes (*branched-from* nodes) at the beginning of branching step π_b.
$DB(\pi_b)$ $\triangleq$ $\{\pi_b^{M_b} \circ l \mid l \in \bar{M}_b\}$
$=$ set of immediate descendants at branching step π_b.
$GS(\pi_b)$ $\triangleq$ set of previously generated nodes at the beginning of branching step π_b.
$AS(\pi_b)$ $\triangleq$ set of currently active nodes at the beginning of branching step π_b.
$ES(\pi_b)$ $\triangleq$ set of all nodes eliminated during branching step π_b.
$U(\pi_b)$ $\triangleq$ cost of upper-bound solution at the beginning of branching step π_b.

With π_t denoting the last branching node before termination of algorithm BB, then:

$BFST$ $\triangleq$ set of all branched-from nodes at the time of termination of BB
$= BFS(\pi_t)$.
$BFST+ = BFST \cup \{\pi_t\}$.
GST $\triangleq$ set of all generated nodes at the time of termination of BB
$= GS(\pi_t)$
$= \{e\} \cup \{\bigcup_{\pi_b \in BFST} DB(\pi_b)\}$.
AST $\triangleq$ set of all currently and previously active nodes at time of termination of BB
$= \bigcup_{\pi_b \in BFST+} AS(\pi_b)$.

Using the above notation, the elimination rules in E can be expressed as one of two types: (1) for each π_z in the set B, eliminate π_z if $U(\pi_b) < L(\pi_z)$, or (2) for each π_z in the set B, eliminate π_z if there exists some π_y in the set A such that $\pi_y \; D \; \pi_z$. With $E_i : A \bullet\!\!\longrightarrow\!\!\bullet B$ denoting a particular rule, the complete set of elimination rules is illustrated diagrammatically in Figure 1.

The general optimum producing branch-and-bound algorithm $BB \triangleq (B_p,S,E,D,L,U)$ is charted in Figure 2 using the D-chart notation [3]. Figure 3 contains a detailed D-chart of a simple implementation of the elimination rules. While other computationally more efficient implementations can be used, the simplicity of the given implementation is conceptually useful.

3. *Proof of Correctness*

In this section several lemmas are proved to establish that the branch-and-bound algorithm (B_p,S,E,D,L,U) generates an optimal solution to the permutation problem (S_n,x,f). The first lemma shows that the set of complete solutions represented by the set of currently active nodes always contains an optimal solution.

LEMMA 1 (Existence of optimal solution in completion of active set). *Given* $BB =$

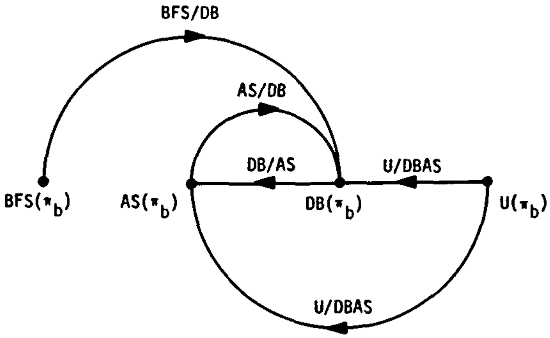

FIG. 1. Diagram of complete set of elimination rules

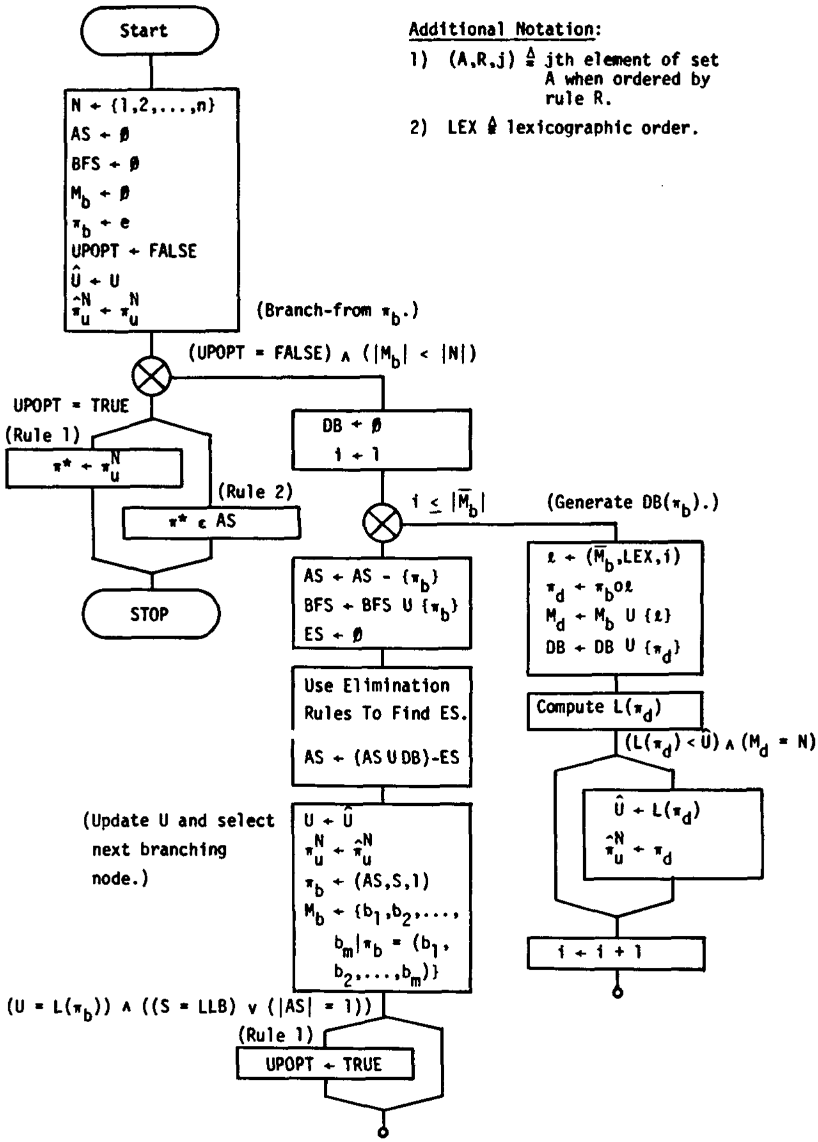

FIG. 2. Chart representation of branch-and-bound algorithm (B_p,S,E,D,L,U)

(B_p,S,E,D,L,U). *Suppose* π_b *is the current branching node. Then the set of complete solutions* $\{AS(\pi_b)\circ\}$ *contains an optimal solution.*

PROOF. The proof is by induction on the active set at successive branching nodes.

Basis. The hypothesis holds trivially for the first branching node $e = \pi^\phi$ since $\{AS(\pi^\phi)\} = \{\pi^\phi\}$ and $\{\pi^\phi\circ\} = S_n$.

Induction Step. The hypothesis is assumed true for branching node π_b and all previous branching nodes. We will show the hypothesis to be true for the next branching node π_c. We can write

$$AS(\pi_c) = \{\pi_y \in AS(\pi_b) \mid \pi_y \neq \pi_b \text{ and } \pi_y \text{ not eliminated by } E\} \cup \{(\pi_b \circ l) \mid l \in \bar{M}_b \text{ and } (\pi_b \circ l) \text{ not eliminated by } E\}. \quad (1)$$

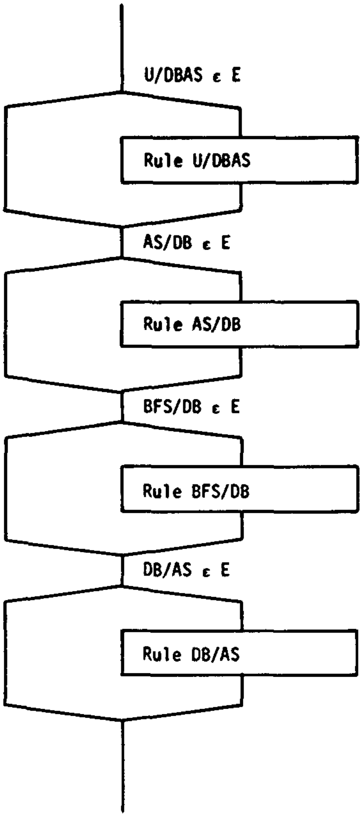

FIG. 3(a). Implementation of elimination rules, Part A

We must show that $\{AS(\pi_c) \circ\}$ contains an optimal solution. Suppose $\pi_y \in AS(\pi_b)$ but $\pi_y \notin AS(\pi_c)$. It follows from (1) that π_y was either eliminated by E or branched-from, i.e. $\pi_y = \pi_b$.

If π_y was eliminated by E then π_y was eliminated with rule $U/DBAS$ or rule DB/AS. If π_y was eliminated with $U/DBAS$, $L(\pi_y) > U$ and no member of $\{\pi_y \circ\}$ is optimal. If π_y was eliminated with DB/AS, then $(\pi_b \circ l)\ D\ \pi_y$ for some $l \in \bar{M}_b$. In this case π_y would be replaced in the active set by $(\pi_b \circ l)$. It follows from the definition of D that $\{(\pi_b \circ l) \circ\}$ contains an optimal solution if $\{\pi_y \circ\}$ does.

Now suppose $\pi_y = \pi_b$. Each immediate descendant of π_b, $\pi_b \circ l$ for $l \in \bar{M}_b$, is in $AS(\pi_c)$ unless it is eliminated by at least one of the following elimination rules in E:

(i) $U/DBAS$. $L((\pi_b \circ l)) > U(\pi_b)$.

(ii) AS/DB. $\pi_a\ D\ (\pi_b \circ l)$, $\pi_a \in AS(\pi_b)$.

(iii) BFS/DB. $\pi_f\ D\ (\pi_b \circ l)$, $\pi_f \in BFS(\pi_b)$.

Rule (i) only eliminates those descendants $(\pi_b \circ l)$ that have no optimal completions. Rules (ii) and (iii) eliminate descendants $(\pi_b \circ l)$ only if there exists another currently active node $\pi_z \in AS(\pi_b) \cap AS(\pi_c)$ such that $\pi_z\ D\ (\pi_b \circ l)$. It follows from the definition of D that $\{\pi_z \circ\}$ contains an optimal solution if $\{(\pi_b \circ l) \circ\}$ does.

We can conclude that for all $\pi_y \in AS(\pi_b)$ there exists some $\pi_z \in AS(\pi_c)$ such that $\pi_z\ \mathfrak{D}\ \pi_y$. Consequently, $\{AS(\pi_c) \circ\}$ contains an optimal solution if $\{AS(\pi_b) \circ\}$ does. □

The next three lemmas demonstrate that (B_p,S,E,D,L,U) as implemented in Figure 2 terminates with an optimal complete solution. Termination occurs when the upper-bound solution has been proved optimal (*Rule* 1), or when the next branching node, π_b, is a complete solution (*Rule* 2). When π_b is a complete solution, it will be shown that the set

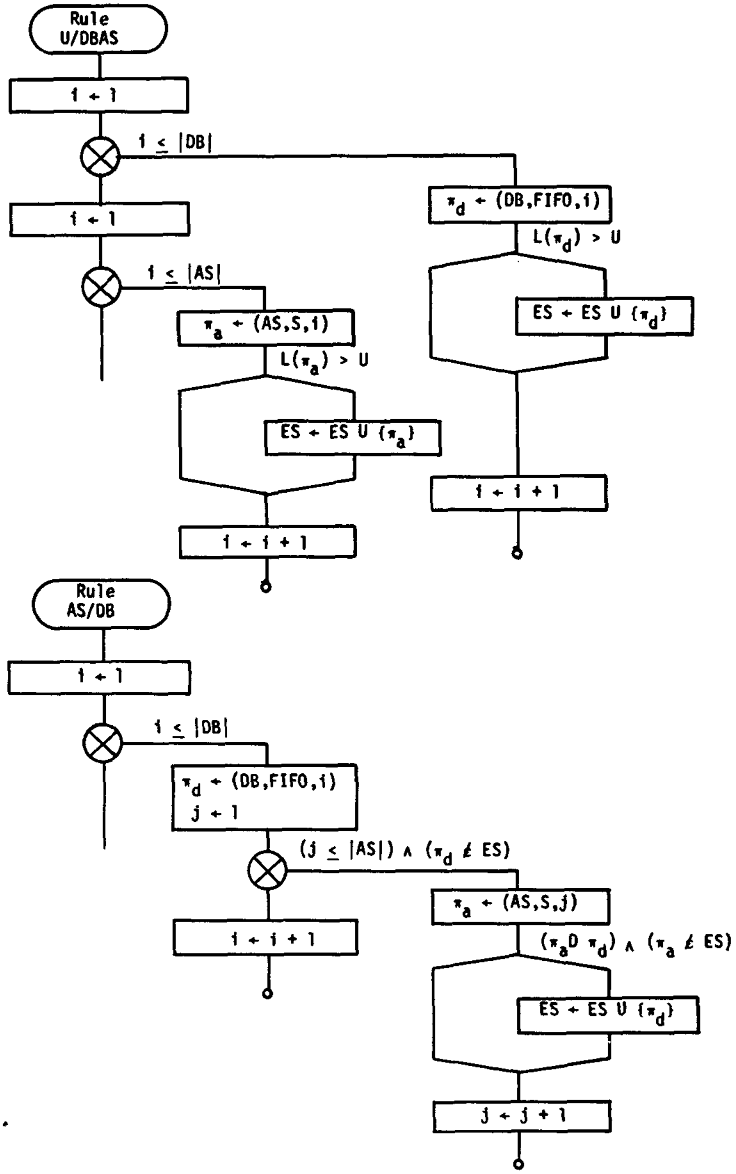

FIG. 3(b). Implementation of elimination rules, Part B

of active nodes, $AS(\pi_b)$, contains an optimal solution, and for the special case when $S = LLB$, π_b is an optimal solution.

LEMMA 2 (Stopping rules for optimal solutions when $S = LLB$). *Given* $BB = (B_p,LLB,E,D,L,U)$. *Let* π_b *be the next branching node.*

(1) *If* $L(\pi_b) = U(\pi_b)$, *then* π_u^N *is an optimal solution* (*Rule* 1).

(2) *If* $M_b = N$, *then* π_b *is an optimal solution* (*Rule* 2).

PROOF. (1) $\pi_b \in AS(\pi_b)$ by definition and $\{AS(\pi_b) \circ\}$ contains an optimal solution (Lemma 1). Since the LLB selection rule chose π_b as the active node with the least lower-bound cost, $L(\pi_b)$ represents a lower bound on the cost of all complete solutions to (S_n,x,f). If $f(\pi_u^N, x) = U(\pi_b) = L(\pi_b)$, then π_u^N must be an optimal solution.

(2) Since π_b is the next branching node under $S = LLB$, $L(\pi_b) = f(\pi_b, x)$ is a lower

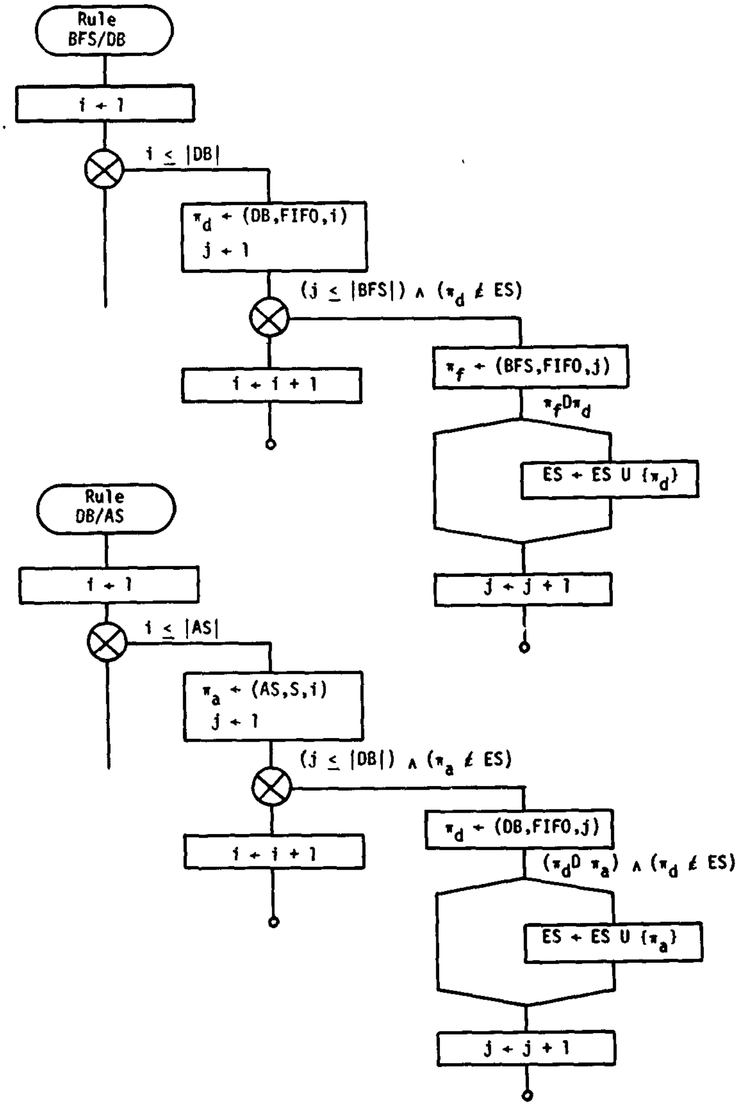

FIG. 3(c). Implementation of elimination rules, Part C

bound on the cost of all solutions in $\{AS(\pi_b)\ \circ\}$. Since $\{AS(\pi_b)\ \circ\}$ contains an optimal solution (Lemma 1), π_b must be one of them. □

LEMMA 3. (Stopping rules for optimal solutions when $S = FIFO$). *Given* $BB = (B_p,FIFO,E,D,L,U)$, *let* π_b *be the next branching node.*

(1) *If* $L(\pi_b) = U(\pi_b)$ *and* $|AS(\pi_b)| = 1$, *then* π_u^N *is an optimal solution* (*Rule* 1).

(2) *If* π_b *is the first branching node such that* $M_b = N$, *then* $AS(\pi_b)$ *consists of complete solutions and some* $\pi_y^N \in AS(\pi_b)$ *is an optimal solution* (*Rule* 2).

PROOF. (1) Since $|AS(\pi_b)| = 1$, $AS(\pi_b) = \{\pi_b\}$. But $\{AS(\pi_b)\ \circ\}$ contains an optimal solution (Lemma 1). Consequently, $L(\pi_b)$ is a lower bound on the cost of an optimal solution. Since π_u^N achieves this bound, $L(\pi_b) = U(\pi_b) = f(\pi_u^N, x)$, π_u^N must be optimal.

(2) Under *FIFO* all active nodes of size k are selected for branching before any active nodes of size $k + 1$. Consequently, if $M_b = N = \{1, 2, \cdots, n\}$, there are no active nodes in $AS(\pi_b)$ of size $n - 1$ or less. Then all nodes in $AS(\pi_b)$ must be complete, $AS(\pi_b) = \{AS(\pi_b)\ \circ\}$, and $AS(\pi_b)$ contains an optimal solution (Lemma 1). □

LEMMA 4. (Stopping rule for optimal solutions when $S = LIFO$). *Given* $BB = (B_p,LIFO,E,D,L,U)$, *let* π_b *be the next branching node.*

(1) *If* $L(\pi_b) = U(\pi_b)$ *and* $|AS(\pi_b)| = 1$, *then* π_u^N *is an optimal solution (Rule 1).*

(2) *If* π_b *is the first branching node such that* $M_b = N$, *then* $AS(\pi_b)$ *consists of complete solutions and some* $\pi_y^N \in AS(\pi_b)$ *is an optimal solution (Rule 2).*

PROOF. (1) Same as for Rule (1) of Lemma 3.

(2) Under $LIFO$ all active nodes of size $k < n$ are selected for branching before any active node of size n. Consequently, if π_b is such that $M_b = N$, $AS(\pi_b)$ must consist of complete solutions, i.e. nodes of size n. As in Lemma 3, $AS(\pi_b) = \{AS(\pi_b) \circ\}$ and it follows from Lemma 1 that $AS(\pi_b)$ contains an optimal solution. □

For the special case when L is defined such that $L(\pi_p) = f(\pi_p \circ l, x)$ if $|\pi_p| = n - 1$, a set of modified selection and stopping rules can be used to terminate the branch-and-bound algorithm with an optimal solution. Lemma 5 describes the minor modifications that would be necessary to implement these rules. Fewer nodes are usually generated when using these modifications but the relative comparisons discussed in Section 4 would remain the same. Attention will consequently be directed at the more general case.

LEMMA 5. (Stopping rules for special lower-bound function). *Given* $BB = (B_p,S,E,D,L,U)$. *Suppose* $L(\pi_p) = f(\pi_p \circ l, x)$ *for all nodes* π_p *with* $|\pi_p| = n - 1$. *(In this case* $\bar{M}_p = \{l\}$.) *Then* π_p *can be interpreted as the unique complete solution* $\pi_p \circ l$. *Let* π_b *be the first branching node such that* $|\pi_b| = n - 1$. *For this case the selection rules and second stopping rule (Rule 2) for an optimal solution can be defined as follows:*

(1) $S = LLB' = LLB$: $\pi_b \circ l$ *is an optimal solution.*

(2) $S = FIFO' = FIFO$: *There exists some* $\pi_a \in AS(\pi_b)$ *such that* $|\pi_a| = n - 1$ *and* $\pi_a \circ k$, $k \in \bar{M}_a$, *is an optimal solution.*

(3) $S = LIFO'$: *(Under* $LIFO'$ *active nodes that represent partial solutions of size* $n - 1$ *are selected for branching only if there are no active nodes of size less than* $n - 1$ *currently active.) There exists some* $\pi_a \in AS(\pi_b)$ *such that* $|\pi_a| = n - 1$ *and* $\pi_a \circ k$, $k \in \bar{M}_a$, *is an optimal solution.*

PROOF. Similar to proofs of Lemmas 2, 3, and 4. □

4. *Theoretical Comparison of Computational Requirements*

In this section the relative computational requirements of $BB_i = (B_p,S,E_i,D_i,L_i,U_i)$, for $i = 1, 2$, will be studied as functions of the parameters E_i, D_i, L_i, and U_i for fixed B and S. The comparisons presented here are valid for any measure of computation that is a monotone nondecreasing function of $|GST|$, $|BFST+|$, $|BFS(\pi_b)|$, and $|AS(\pi_b)|$, $\forall \pi_b \in BFST+$. This generalized case is important when E contains elimination rules that search sets $BFS(\pi_b)$ and $AS(\pi_b)$ to check for dominance. However, to simplify the exposition the total number of nodes generated, $|GST_i|$, will be used as a relative measure of required execution time, and the maximum size of the active node set, $\max_{\pi_y \in BFST_i+} |AS_i(\pi_y)|$, will be used as a relative measure of the required storage. When comparing parameters the following definitions will be used:

(i) $D_2 \subseteq D_1$ if for every pair of partial solutions π_y and π_z, $\pi_y D_2 \pi_z$ only if $\pi_y D_1 \pi_z$.

(ii) $L_2 \leq L_1$ if for every partial solution π_y, $L_2(\pi_y) \leq L_1(\pi_y)$.

It follows directly that $|GST_1| \leq |GST_2|$ if $BFST_1 \subseteq BFST_2$. It will usually be easier to prove $BFST_1 \subseteq BFST_2$ and let $|GST_1| \leq |GST_2|$ follow as a consequence.

The first theorem in this section shows that the required storage and execution time cannot be increased by adding node elimination rule $U/DBAS$ to E.

THEOREM 1. (The set of branched-from nodes and the maximum number of active nodes cannot increase when nodes are eliminated on the basis of their lower-bound cost exceeding an upper-bound cost). *Given* $BB_1 = (B_p,S,E_1,D,L,U)$ *and* $BB_2 = (B_p,S,E_2,D,L,U)$. *If* $E_1 = E_2 \cup \{U/DBAS\}$ *then*

(1) $BFST_1 \subseteq BFST_2$, *and*

(2) $\max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| \leq \max_{\pi_y \in BFST_2+} |AS_2(\pi_y)|$.

PROOF. Rule $U/DBAS$ eliminates those nodes $\pi_y \in DB(\pi_b) \cup AS(\pi_b)$, $\pi_b \in BFST$, with $L(\pi_y) > U(\pi_b) \geq f(\pi^*, x)$. The important fact is that when $\pi_y\, D\, \pi_z$ and $U/DBAS$ eliminates π_y, then $U/DBAS$ will also eliminate π_z. This follows from the requirement on D that $\pi_y\, D\, \pi_z$ only if $L(\pi_y) \leq L(\pi_z)$. Another property of the lower-bound function is that $L(\pi_d) \geq L(\pi_y)$ when π_d is a descendant of π_y. Consequetly, $U/DBAS$ eliminates all nodes previously eliminated by descendants of π_y using dominance function D. That is, when $\pi_z\, D\, \pi_w$, then $L(\pi_w) \geq L(\pi_z) \geq L(\pi_y) > U(\pi_b) \geq f(\pi^*, x)$ and $U/DBAS$ eliminates π_w.

We can conclude that when π_y is eliminated under BB_2, then π_y is also eliminated under BB_1 if it is generated. Furthermore, $U/DBAS$ cannot increase the size of the currently active set because $U/DBAS$ eliminates nodes before they become active. (1) and (2) follow directly. □

While the above theorem justifies our intuitive feeling about using elimination rule $U/DBAS$, a set of counterexamples can be constructed to contradict our intuition about the universal value of a stronger dominance function. In particular, we conclude that the computational requirements of (B_p,S,E,D,L,U) are not necessarily a monotone nonincreasing function of the dominance function D. We give here a counterexample for the case when $S = LLB$. Similar counterexamples can be constructed for $S = FIFO$ and $LIFO$.

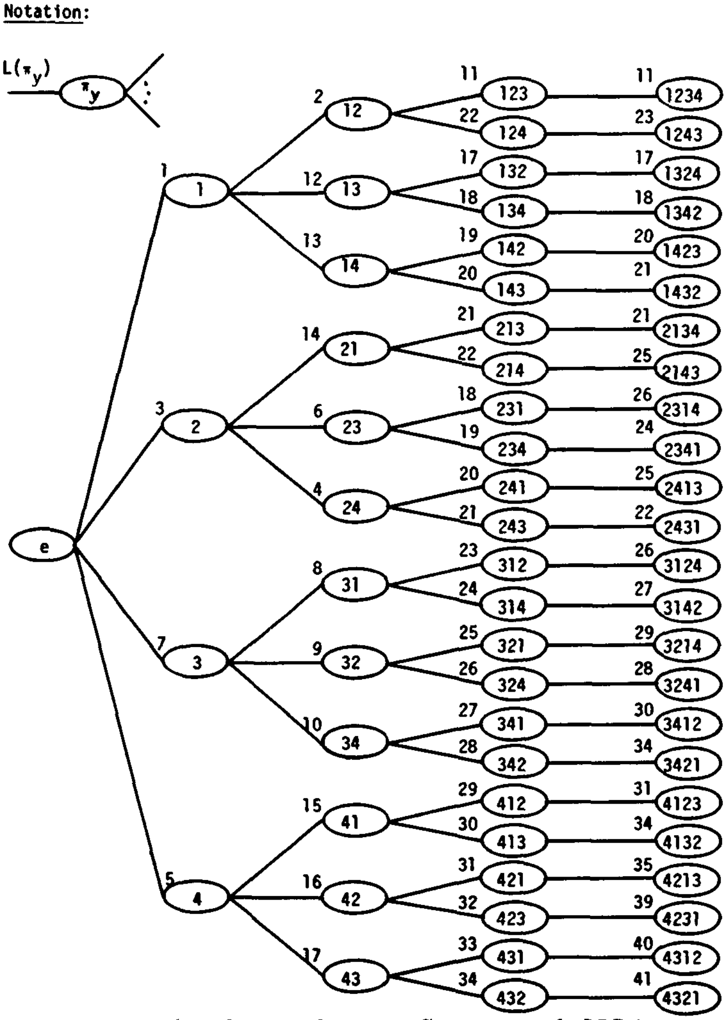

FIG. 4. Complete search tree for Counterexample LLB-1

COUNTEREXAMPLE *LLB*-1. (When using the *LLB* selection rule, the total number of nodes generated and the maximum number of active nodes are not necessarily monotone functions of the dominance function). *Given* $BB_1 = (B_p,LLB,E,D_1,L,U)$ *and* $BB_2 = B_p,LLB,E,D_2,L,U)$. *If* $D_2 \subseteq D_1$ *it does not necessarily follow that*

(1) $|GST_1| \leq |GST_2|$, *or*

(2) $max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| \leq max_{\pi_y \in BFST_2+} |AS_2(\pi_y)|$.

PROOF. The problem example is defined by the complete search tree of Figure 4. The lower-bound cost associated with node π_y, $L(\pi_y)$, is written to the upper left of the corresponding node in the tree. For ease of notation $\pi_a \triangleq (a_1, a_2, \cdots, a_m)$ will be denoted by $a_1a_2 \cdots a_m$, and membership in the dominance function D_i, $\pi_y D_i \pi_z$, will be denoted by the pair (π_y, π_z). In particular, let $BB_1 = (B_p,LLB,\{DB/AS\},D_1,L,\infty)$ and $BB_2 = (B_p,LLB,\{DB/AS\},D_2,L,\infty)$ where $D_2 = \{(24,4), (23,3)\} \cup \{(\pi_y, \pi_z) \mid L(\pi_y) \leq L(\pi_z)$ and $M_y = N\}$ and $D_1 = D_2 \cup \{(12, 2)\}$. $U_i = \infty$ is used to indicate that no solution is known at the start of algorithm BB_i, and $U_i(e)$ is set equal to some number guaranteed to be greater than $f(\pi, x)$, $\forall \pi \in S_n$.

The steps of BB_i will be described in terms of ordered sets $(AS_i(\pi_b), LLB, j)$, $(DB_i(\pi_b), LEX, k)$, and $(ES_i(\pi_b), FIFO, l,)$, where (A,R,j) is defined as the jth element of set A when ordered by Rule R. The steps of BB_1 are as follows:

BB_1

π_b	$(AS_1(\pi_b), LLB, j)$	$(DB_1(\pi_b), LEX, k)$	$(ES_1(\pi_b), FIFO, l)$
e	(e)	(1, 2, 3, 4)	ϕ
1	(1, 2, 4, 3)	(12, 13, 14)	(2)
12	(12, 4, 3, 13, 14)	(123, 124)	ϕ
4	(4, 3, 123, 13, 14, 124)	(41, 42, 43)	ϕ
3	(3, 123, 13, 14, 41, 42, 43, 124)	(31, 32, 34)	ϕ
31	(31, 32, 34, 123, 13, 14, 41, 42, 43, 124)	(312, 314)	ϕ
32	(32, 34, 123, 13, 14, 41, 42, 43, 124, 312, 314)	(321, 324)	ϕ
34	(34, 123, 13, 14, 41, 42, 43, 124, 312, 314, 321, 324)	(341, 342)	ϕ
123	(123, 13, 14, 41, 42, 43, 124, 312, 314, 321, 324, 341, 342)	(1234)	(13, 14, 41, 42, 43, 124, 312, 314, 321, 324, 341, 342)
1234	(1234)		
Rule 2 $\Rightarrow \pi^* = 1234$			

The statistics for BB_1 are $|GST_1| = 23$ and $\max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| = |AS_1(123)| = 13$.

The steps for BB_2 are as follows:

BB_2

π_b	$(AS_2(\pi_b), LLB, j)$	$(DB_2(\pi_b), LEX, k)$	$(ES_2(\pi_b), FIFO, l)$
e	(e)	(1, 2, 3, 4)	ϕ
1	(1, 2, 4, 3)	(12, 13, 14)	ϕ
12	(12, 2, 4, 3, 13, 14)	(123, 124)	ϕ
2	(2, 4, 3, 123, 13, 14, 124)	(21, 23, 24)	(3, 4)
24	(24, 23, 123, 13, 14, 21, 124)	(241, 243)	ϕ
23	(23, 123, 13, 14, 21, 241, 243, 124)	(231, 234)	ϕ
123	(123, 13, 14, 21, 231, 234, 241, 243, 124)	(1234)	(13, 14, 21, 231, 234, 241, 243, 124)
1234	(1234)		
Rule 2 $\Rightarrow \pi^* = 1234$			

In the case of BB_2, $|GST_2| = 18$ and $\max_{\pi_y \in BFST_2+} |AS_2(\pi_y)| = |AS_2(123)| = 9$. □

The next theorem will show that the computational requirements of (B_p,S,E,D,L,U) when $S = FIFO$ or $LIFO$ are monotone functions of both the lower-bound function L and the initial upper-bound cost $U(e)$. Corollary 1 establishes that the relationship is valid for any measure of computation that is a monotone nondecreasing function of $|GST|$, $|BFST+|$, $|BFS(\pi_b)|$, and $|AS(\pi_b)|$, $\forall \pi_b \in BFST+$.

THEOREM 2 (When using the *FIFO* or *LIFO* selection rule, the set of branched-from nodes and the maximum number of active nodes are monotone functions of the lower-bound function and the initial upper-bound cost). *Given* $BB_1 = (B_p,S_1,E,D,L_1,U_1)$ *and* $BB_2 = (B_p,S_2,E,D,L_2,U_2)$. *If* $S_1 = S_2 = FIFO$ *or* $LIFO$, $L_1 \geq L_2$, *and* $U_1(e) \leq U_2(e)$, *then*

(1) $BFST_1 \subseteq BFST_2$, *and*

(2) $max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| \leq max_{\pi_y \in BFST_2+} |AS_2(\pi_y)|$.

PROOF.[1] First assume $U/DBAS \notin E$ and let π_t be the branching node at the time of termination of BB_1. We will prove by induction that

(i) $BFS_1(\pi_b) = BFS_2(\pi_b)$, $\forall \pi_b \in BFS_1(\pi_t) \cup \{\pi_t\} = BFST_1+$.

(ii) $AS_1(\pi_b) = AS_2(\pi_b)$, $\forall \pi_b \in BFST_1+$.

(iii) $U_1(\pi_b) \leq U_2(\pi_b)$, $\forall \pi_b \in BFST_1+$.

Basis. (i)–(iii) are true for the first branching node $\pi_b = e$.

Induction step. Let π_k denote the kth potential branching node when the complete permutation search tree is enumerated using rule $S = S_1 = S_2$. If node π_k or one of its ancestors has already been eliminated, then π_k is not active and the next node, π_{k+1}, is enumerated. If π_k is active, then branching and updating proceeds as in the usual branch-and-bound algorithm. This scheme coincides with the usual branch-and-bound algorithm at all branched-from nodes $\pi_b \in BFST_i$, $i = 1, 2$. We will prove (i)–(iii) by induction on successive candidate nodes π_k until termination of BB_1. Assume

(i) $BFS_1(\pi_k) = BFS_2(\pi_k)$

(ii) $AS_1(\pi_k) = AS_2(\pi_k)$, and

(iii) $U_1(\pi_k) \leq U_2(\pi_k)$.

The inductive step can then be verified in each of the following cases:

(a) $\pi_k \in BFST_1 \cap BFST_2$,

(b) $\pi_k \in BFST_1 \cap \overline{BFST_2}$,

(c) $\pi_k \in \overline{BFST_1} \cap BFST_2$, and

(d) $\pi_k \in \overline{BFST_1} \cap \overline{BFST_2}$.

Next assume $U/DBAS \in E$ and again let π_t denote the branching node at the time of termination of BB_1. It can then be shown by induction, in a manner analogous to the above, that

(i) $BFS_1(\pi_b) \subseteq BFS_2(\pi_b)$, $\forall \pi_b \in BFS_1(\pi_t) \cup \{\pi_t\} = BFST_1+$,

(ii) If $\pi_y \in BFS_2(\pi_b) - BFS_1(\pi_b)$, $\pi_b \in BFST_1+$, then $L_1(\pi_y) > U_1(\pi_b)$,

(iii) $AS_1(\pi_b) \subseteq AS_2(\pi_b)$, $\forall \pi_b \in BFST_1+$,

(iv) If $\pi_y \in AS_2(\pi_b) - AS_1(\pi_b)$, $\pi_b \in BFST_1+$, then $L_1(\pi_y) > U_1(\pi_b)$, and

(v) $U_1(\pi_b) \leq U_2(\pi_b)$, $\forall \pi_b \in BFST_1+$. □

COROLLARY 1. (Generalization of Theorem 2). *Given* $BB_1 = (B_p,S_1,E,D,L_1,U_1)$ *and* $BB_2 = (B_p,S_2,E,D,L_2,U_2)$. If $S_1 = S_2 = FIFO$ *or* $LIFO$, $L_1 \geq L_2$, *and* $U_1(e) \leq U_2(e)$, *then*

(1) $BFST_1+ \subseteq BFST_2+$,

and for all $\pi_b \in BFST_1+$,

(2) $BFS_1(\pi_b) \subseteq BFS_2(\pi_b)$,

(3) *if* $\pi_y \in BFS_2(\pi_b) - BFS_1(\pi_b)$, *then* $L_1(\pi_y) > U_1(\pi_b)$,

(4) $AS_1(\pi_b) \subseteq AS_2(\pi_b)$,

(5) *if* $\pi_y \in AS_2(\pi_b) - AS_1(\pi_b)$, *then* $L_1(\pi_y) > U_1(\pi_b)$, *and*

(6) $U_1(\pi_b) \leq U_2(\pi_b)$.

PROOF. (1)–(6) were proved in Theorem 2. □

[1] This proof is lengthy, and only an outline is given here. For details, see [9].

Theorem 3 and Counterexample LLB-2 will now show that when $S = LLB$, the computational requirements are monotone functions of the initial upper-bound cost $U(e)$ but not necessarily the lower-bound function L. This counter-intuitive observation is related to the fact that modifying the lower-bound function may also change the branching order. Corollary 2 indicates the general version of Theorem 3 and Lemma 6 will be used in the proof.

LEMMA 6. *Given* (B_p,LLB,E,D,L,U). *Let* π^* *be an optimal complete solution. If* π_y *is any partial solution such that* $L(\pi_y) > f(\pi^*, x)$, *then*

(1) $\pi_y \notin BFST$, *and*

(2) $\pi_y \not D\ \pi_z$ *if* $\pi_z \in BFST$.

PROOF. Part (1) is proved by contradiction. Assume $\pi_y \in BFST$. Then it follows from branching rule LLB that $L(\pi_y) \leq f(\pi^*, x)$, which is a contradiction.

Part (2) is also proved by contradiction. Assume $\pi_y\, D\, \pi_z$ for some $\pi_z \in BFST$. It follows from the consistency requirement on D that $L(\pi_y) \leq L(\pi_z)$. From part (1), $\pi_z \in BFST$ only if $L(\pi_z) \leq f(\pi^*, x)$. Consequently, $L(\pi_y) \leq L(\pi_z) \leq f(\pi^*, x)$. Again this contradicts the assumption that $L(\pi_y) > f(\pi^*, x)$. □

THEOREM 3. (When using the LLB selection rule, the set of branched-from nodes and the maximum number of active nodes are monotone functions of the initial upper-bound cost $U(e)$). *Given* $BB_1 = (B_p,LLB,E,D,L,U_1)$ *and* $BB_2 = (B_p,LLB,E,D,L,U_2)$. *If* $U_1(e) \leq U_2(e)$, *then*

(1) $BFST_1 \subseteq BFST_2$, *and*

(2) $max_{\pi_y \in BFST_1+} | AS_1(\pi_y) | \leq max_{\pi_y \in BFST_2+} | AS_2(\pi_y) |$.

PROOF. Since $L_1 = L_2 = L$, $D_1 = D_2 = D$, and $E_1 = E_2 = E$, the order of branching under $S = LLB_{LIFO}$ or LLB_{FIFO} is independent of the upper-bound cost $U_i(\pi_b)$. The upper bound is used in two ways, neither of which modifies the order of branching: (a) it is used with the first stopping rule (Rule 1) to terminate BB_i if $L(\pi_b) = U_i(\pi_b)$ at branching node π_b, and (b) it is used with elimination rule $U/DBAS$ to eliminate all nodes $\pi_q \in DB(\pi_b) \cup AS(\pi_b)$ such that $L_i(\pi_q) > U_i(\pi_b)$. But we know from Lemma 6 that π_q is never branched-from under $S = LLB$ if $L_i(\pi_q) \geq f(\pi^*, x)$. Consequently, the upper-bound cost cannot modify the order of branching.

Now assume $U/DBAS \notin E$ and let π_t be the branching node at the time of termination of BB_1. Using the same argument as in Theorem 2 with $L = L_1 = L_2$, we can prove by induction on successive branching nodes that

(i) $BFS_1(\pi_b) = BFS_2(\pi_b)$, $\forall \pi_b \in BFS_1(\pi_t) \cup \{\pi_t\} = BFST_1+$,

(ii) $AS_1(\pi_b) = AS_2(\pi_b)$, $\forall \pi_b \in BFST_1+$, and

(iii) $U_1(\pi_b) \leq U_2(\pi_b)$, $\forall \pi_b \in BFST_1+$.

Next assume $U/DBAS \in E$ and let π_t be the branching node at the time of termination of BB_1. Again using an argument similar to the one in Theorem 2 with $L = L_1 = L_2$, we can prove by induction on successive branching nodes that

(i) $BFS_1(\pi_b) = BFS_2(\pi_b)$, $\forall \pi_b \in BFST_1+$,

(ii) $AS_1(\pi_b) \subseteq AS_2(\pi_b)$, $\forall \pi_b \in BFST_1+$,

(iii) if $\pi_y \in AS_2(\pi_b) - AS_1(\pi_b)$, then $L(\pi_y) > U_1(\pi_b)$, and

(iv) $U_1(\pi_b) \leq U_2(\pi_b)$, $\forall \pi_b \in BFST_1+$.

In both cases $BFS_1(\pi_t) \cup \{\pi_t\} = BFST_1+ \subseteq BFS_2(\pi_t) \cup \{\pi_t\} \subseteq BFST_2+$, and (1) and (2) follow directly. □

COROLLARY 2. (Generalization of Theorem 3). *Given* $BB_1 = (B_p,LLB,E,D,L,U_1)$ *and* $BB_2 = (B_p,LLB,E,D,L,U_2)$. *If* $U_1(e) \leq U_2(e)$, *then*

(1) $BFST_1+ \subseteq BFST_2+$,

and for all $\pi_b \in BFST_1+$,

(2) $BFS_1(\pi_b) = BFS_2(\pi_b)$,

(3) $AS_1(\pi_b) \subseteq AS_2(\pi_b)$,

(4) *if* $\pi_y \in AS_2(\pi_b) - AS_1(\pi_b)$, *then* $L(\pi_y) > U_1(\pi_b)$, *and*

(5) $U_1(\pi_b) \leq U_2(\pi_b)$.

PROOF. (1)–(5) follow from the proof of Theorem 3. □

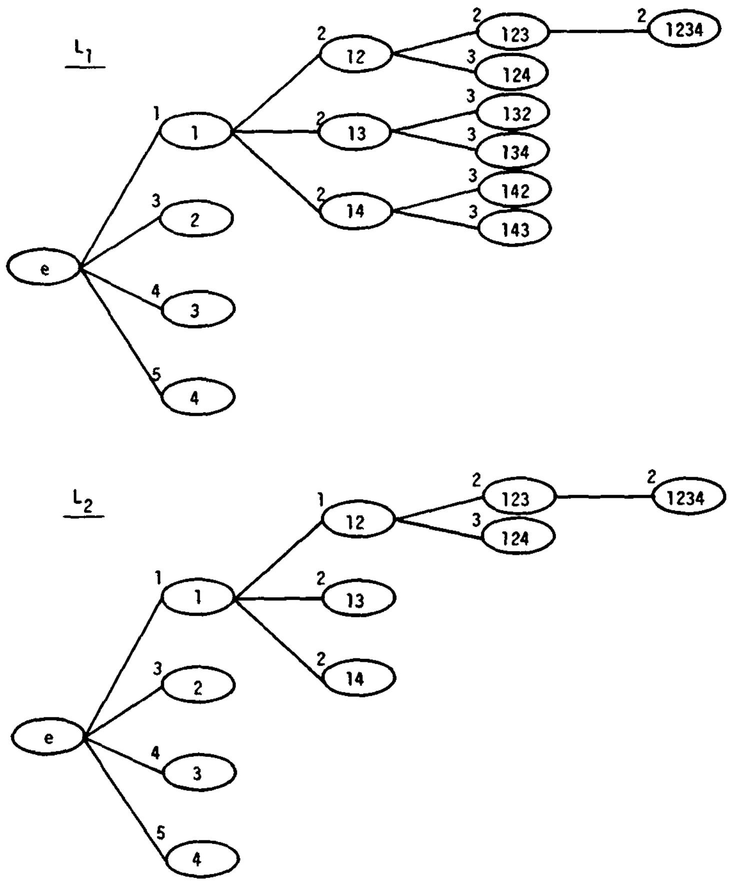

FIG. 5. Partial search trees for Counterexample *LLB*-2

COUNTEREXAMPLE *LLB*-2 (When using the *LLB* selection rule, the set of branched-from nodes and the maximum number of active nodes are not necessarily monotone functions of the lower-bound function). *Given* $BB_1 = (B_p,LLB,E,D,L_1,U)$ *and* $BB_2 = (B_p,LLB,E,D,L_2,U)$. *If* $L_1 \geq L_2$ *it does not necessarily follow that*

(1) $| GST_1 | \leq | GST_2 |$, *or*

(2) $max_{\pi_y \in BFST_1+} | AS_1(\pi_y) | \leq max_{\pi_y \in BFST_2+} | AS_2(\pi_y) |$.

PROOF. Let $BB_1 = (B_p,LLB_{LIFO},\phi,\phi,L_1,U)$ and $BB_2 = (B_p,LLB_{LIFO},\phi,\phi,L_2,U)$. Lower-bound functions L_1 and L_2 used for this example are defined by the partial search trees of Figure 5.

Algorithm BB_1 executes the following steps:

BB_1

π_b	$(AS_1(\pi_b), LLB_{LIFO}, j)$	$(DB_1(\pi_b), LEX, k)$	$(ES_1(\pi_b), FIFO, l)$
e	(e)	(1, 2, 3, 4)	ϕ
1	(1, 2, 3, 4)	(12, 13, 14)	ϕ
14	(14, 13, 12, 2, 3, 4)	(142, 143)	ϕ
13	(13, 12, 143, 142, 2, 3, 4)	(132, 134)	ϕ
12	(12, 134, 132, 143, 142, 2, 3, 4)	(123, 124)	ϕ
123	(123, 124, 134, 132, 143, 142, 2, 3, 4)	(1234)	ϕ
1234	(1234, 124, 134, 132, 143, 142, 2, 3, 4)		
Rule 1 $\Rightarrow \pi^* = 1234$			

The computational requirements of BB_1 are $|GST_1| = 15$ and $\max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| = 9$.

Algorithm BB_2 executes the following steps:

BB_2

π_b	$(AS_2(\pi_b), LLB_{LIFO}, j)$	$(DB_2(\pi_b), LEX, k)$	$(ES_1(\pi_b), FIFO, l)$
e	(e)	(1, 2, 3, 4)	ϕ
1	(1, 2, 3, 4)	(12, 13, 14)	ϕ
12	(12, 14, 13, 2, 3, 4)	(123, 124)	ϕ
123	(123, 14, 13, 124, 2, 3, 4)	(1234)	ϕ
1234	(1234, 14, 13, 124, 2, 3, 4)		
Rule 1 $\Rightarrow \pi^* = 1234$			

The computational requirements of BB_2 and $|GST_2| = 11$ and $\max_{\pi_y \in BFST_2+} |AS_2(\pi_y)| = 7$. □

5. Conclusions

We have proposed the sextuple characterization scheme (B,S,E,D,L,U) for branch-and-bound algorithms and demonstrated how it can be used to describe the common optimum producing algorithms for permutation problems. This framework provided a sufficient basis for a theoretical investigation of the relative computational requirements of various algorithms for a general class of problems. Figure 6 summarizes these results. Theorem 1 shows that you cannot lose by eliminating those currently active and newly generated nodes that exceed an upper-bound solution cost. Theorems 2 and 3 show that you cannot lose by using a better solution as the initial upper bound. Computational experience with a wide variety of flow-shop problems in fact demonstrates that total computation can be significantly reduced by using a good heuristic method to obtain an initial upper-bound solution, and then using an upper-bound dominance function like $U/DBAS$ to eliminate suboptimal solutions from further consideration [9]. The fact that the number of generated and active nodes are not necessarily monotone nonincreasing functions of the dominance and lower-bound functions does not mean that stronger dominance and lower-bound functions are worthless. It just says that the number of generated and active nodes may sometimes increase. In practice one usually finds a decrease, but the total computation time may actually increase if the time required to compute the stronger D and L exceeds the savings.

Our analysis can be extended and related to other approaches by choosing new special cases of the sextuple parameters. For example, when $B \triangleq B_p$, $S \triangleq FIFO$, $E \triangleq \{AS/DB, DB/AS\}$, D and L can be defined [9] so that (B,S,E,D,L,U) describes the dynamic programming approach to one-machine sequencing problems [6].

$BB_i = (B_p,S,E_i,D_i,L_i,U_i)$

Requirements	$E_1 = E_2 \cup \{U/DBAS\}$	$D_1 \supseteq D_2$	$L_1 \geq L_2$	$U_1(e) \leq U_2(e)$
(1) $BFST_1 \subseteq BFST_2$, (2) $|GST_1| \leq |GST_2|$, and (3) $\max_{\pi_y \in BFST_1+} |AS_1(\pi_y)| \leq \max_{\pi_y \in BFST_2+} |AS_2(\pi_y)|$	True (Theorem 1)	False (counter-examples LLB-1, $FIFO$-1, $LIFO$-1)	$S = LLB \Rightarrow$ False (counter-example LLB-2); $S = FIFO$ or $LIFO \Rightarrow$ True (Theorem 2)	True (Theorems 2 and 3)

FIG. 6. Relative computational requirements as functions of E, D, L, and U for fixed B_p and S

REFERENCES

1. AGIN, N. Optimum seeking with branch and bound. *Manag. Sci. 13*, 4 (Dec. 1966), B 176–B 185.
2. BALAS, E. A note on the branch-and-bound principle. *Oper. Res. 16*, 2 (March–April 1968), 442–444; Errata, *Oper. Res. 16*, 4 (1968), 886.
3. BRUNO, J., AND STEIGLITZ, K. The expression of algorithms by charts. *J. ACM 19*, 3 (July 1972), 517–525.
4. FOX, B. L., AND SCHRAGE, L. E. The value of various strategies in branch-and-bound. Unpublished rep., June 1972.
5. GEOFFRION, A. M., AND MARSTEN, R. E. Integer programming algorithms: A framework and state of the art survey. *Manag. Sci. 18*, 9 (May 1972), 465–491.
6. HELD, M., AND KARP, R. A dynamic programming approach to sequencing problems. *J. SIAM 10*, 1 (March 1962), 196–210.
7. HELD, M., AND KARP, R. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming 1*, 1 (Oct. 1971), 6–25.
8. IGNALL, E., AND SCHRAGE, L. Application of branch and bound technique to some flow-shop scheduling problems. *Oper. Res. 13*, 3 (May–June 1965), 400–412.
9. KOHLER, W. H. Exact and approximate algorithms for permutation problems. Ph.D. Diss., Princeton U., Princeton, N.J., 1972.
10. LAWLER, E. L., AND WOOD, D. E. Branch-and-bound methods: A survey. *Oper. Res. 14*, 4 (July–Aug. 1966), 699–719.
11. MITTEN, L. G. Branch-and-bound methods: General formulation and properties. *Oper. Res. 18*, 1 (Jan.–Feb. 1970), 24–34.
12. NOLLEMEIER, H. Ein branch-and-bound-verfahren-generator. *Computing 8* (1971), 99–106.
13. PIERCE, J. F., AND CROWSTON, W. B. Tree-search algorithms for quadratic assignment problems. *Naval Res. Logistics Quart. 18*, 1 (March 1971), 1–36.
14. ROY, B. Procedure d'exploration par séparation et évaluation. *Rev. Francaise d'Informatique et de Recherche Opérationelle*, 6 (1969), 61–90.
15. SCHRAGE, L. Solving resource-constrained network problems by implicit enumeration—nonpreemptive case. *Oper. Res. 18*, 2 (March–April 1970), 263–278.
16. SCHRAGE, L. Solving resource-constrained network problems by implicit enumeration—preemptive case. *Oper. Res. 20*, 3 (May–June 1972), 668–677.

RECEIVED SEPTEMBER 1972; REVISED FEBRUARY 1973