Journal of the Operations Research
Society of Japan
Vol. 20, No. 1, March, 1977

ON THE COMPUTATIONAL EFFICIENCY OF BRANCH-AND-BOUND ALGORITHMS

TOSHIHIDE IBARAKI, *Kyoto University*

(Received June 4, 1976; Revised September 16, 1976)

Abstract. The behavior of the number of partial problems T(A) which are decomposed in a branch-and-bound algorithm A (T(A) may be taken as a measure for the computational efficiency of A) is investigated in a fairly general setting. The first result is that the mean number $\bar{T}(n)$ of T(A) when A is applied to problems of size n grows at least as fast as exponentially with n, under relatively mild conditions, if A uses only the lower bound test as most of the conventional branch-and-bound algorithms do. Then it is pointed out that a possible way to avoid this exponential growth is to use the dominance test together with the lower bound test. The dominance test is also interesting from the view point of unifying a wide variety of algorithms as branch-and-bound.

These points are exemplified by the well known Dijkstra algorithm for the shortest path problem and the Johnson algorithm for the two-machine flow-shop scheduling problem, for which $\bar{T}(n) \leq n-1$ holds by virtue of the dominance test.

1. Introduction

Branch-and-bound is a computational principle used to solve various combinatorial optimization problems, in particular those which are not so nicely structured as to permit very efficient algorithms. For many "difficult" problems, such as the integer programming problem, the traveling salesman problem and various scheduling problems, branch-and-bound is reportedly the only practical approach.

As is well known, e.g. [1, 10, 11, 21, 23, 25, 27], the underlying idea of branch-and-bound is to decompose a given minimization problem P_0 into more than one partial problem of smaller size. The decomposition is repeatedly applied to the generated partial problems until each undecomposed one is either solved or proved not to yield an optimal solution of P_0. For theo-

retical simplicity, the computational efficiency of a branch-and-bound algorithm A is commonly measured by the number $T(A)$ of partial problems which are decomposed in the entire computation (e.g., [14, 21, 27, 28]). This measure is also adopted in this paper. The actual time required to carry out the entire computation of A is roughly given by $T(A)\cdot\bar{t}$, where $\bar{t}$ is the average time required to solve a partial problem. Thus, though smaller $T(A)$ does not always imply shorter computation time (since $\bar{t}$ may increase when we make $T(A)$ small), $T(A)$ conveys essential information on the total computation time. For example, if $T(A)$ grows exponentially with the size n of given problems, the total computation time also increases at least exponentially as a function of n (under a quite reasonable assumption that $\bar{t}$ is a nondecreasing function of n).

In order to make $T(A)$ small, it is crucial how to test given partial problems and, if possible, to conclude the impossibility of yielding an optimal solution of P_0. Two types of tests, the *lower bound test* (e.g., [1, 23, 25, 27]) and the *dominance test* (e.g., [21, 15]), are known. The first one is most common and is done by calculating a lower bound $g(P_i)$ of the optimal value $f(P_i)$ of a partial problem P_i; if $g(P_i) > z$, where z is the value of the best feasible solution of P_0 obtained so far, we conclude that P_i does not provide an optimal solution of P_0 and P_i is terminated (excluded from consideration). The second one is based on a binary relation D, called a dominance relation, such that P_iDP_j implies $f(P_i) < f(P_j)$; thus P_j is terminated if some P_i satisfying P_iDP_j has already been generated. Most of the current branch-and-bound algorithms are using only the lower bound test, but there is a tendency of the increasing use of the dominance test (e.g.,[15, 16, 22, 29]). [15] contains additional references discussing practical uses of dominance relations.

In spite of the considerable success of branch-and-bound algorithms, it is empirically known that $T(A)$ usually increases exponentially with the size of the given problem. In some cases, this is mathematically proved (e.g.,[17], Section 3.11 of [27]). The first purpose of this paper is to show the exponential growth of $T(A)$, when only the lower bound test is used, in a rather general mathematical setting if certain assumptions are satisfied. It says that $\bar{T}(n)$, the expected value of $T(A)$ when A is applied to a set of problems with size n, grows at least as fast as exponentially with n. To obtain this result, only a mild assumption is made on branch-and-bound algorithm A and problems to be solved. The assumption states, loosely speaking, that a branch-and-bound algorithm with the lower bound test only cannot avoid its

exponential growth of $T(A)$ unless either given problems are so simple as to generate the number of partial problems less than exponential even if all possible decompositions are performed, or an exceptionally accurate lower bounding function g is available such that $g(P_i)$ is very close to $f(P_i)$ for all partial problems P_i but for a small number of exceptions. In some cases, the exponential growth of $\overline{T}(n)$ can actually be proved from the above result. Section 4 contains such examples.

Note here that the branch-and-bound principle is general enough to accept the problems which are known to be NP-complete or more difficult. (According to Cook [4] and Karp [19], the NP-completeness is considered as strong evidence for the nonexistence of a polynomial time algorithm.) Therefore it is unreasonable to expect that $T(A)$ of any branch-and-bound algorithm can be made to grow less than exponentially by means of a lower bounding function g which is easily calculated (say, in time bounded by a polynomial of size n). The above result, however, is stronger in that $T(A)$ grows exponentially even for those problems much easier than NP-complete, and in that the expected value of $T(A)$ grows exponentially with n as opposed to the worst case result considered in the discussion of NP-completeness. This may suggest that branch-and-bound with only the lower bound test is not powerful enough to exploit all aspects of the problem structure useful to improve the computational efficiency.

It is then shown in the latter half of this paper that a possible way to overcome this difficulty is to use an appropriate dominance test. Under certain assumptions, a dominance relation D can restrict the set of partial problems, which are possibly decomposed, from $\mathcal{P}$ (the set of all possible partial problems) to $\mathcal{P}_D$ always satisfying $\mathcal{P}_D \subset \mathcal{P}$, where $\mathcal{P}_D$ is determined independently of lower bounding function g. Thus D has an effect of reducing the size of the set of partial problems which undergo the lower bound test. In particular, if the size of $\mathcal{P}_D$ grows less than exponentially, so does the $T(A)$ of the resulting algorithm.

As such examples, the well known Dijkstra algorithm [5] for the shortest path problem and the Johnson algorithm [18] for the two-machine flow-shop scheduling problem are analyzed and shown that $\overline{T}(n) \leq n-1$, while $\overline{T}(A)$ grows exponentially without the dominance test (under certain assumptions). It may also be interesting to see the fact itself that the Dijkstra algorithm and the Johnson algorithm can be viewed as branch-and-bound algorithms. This may exhibit another aspect of the power of the dominance test; it helps us unify a wide variety of algorithms under the name of branch-and-bound. Thus it is the second purpose of this paper to show theoretically (and at the same time

tutorially) that the dominance test is a quite natural and powerful tool which improves the computational efficiency for most of the existing branch-and-bound algorithms.

2. Branch-and-Bound Algorithm

This section gives a formal description of a branch-and-bound algorithm to solve a minimization problem P_0. The motivation for various concepts introduced in this section and the validity of the resulting algorithm may be found in papers such as [1, 10, 11, 14, 15, 21, 23, 25, 27].

The way in which the original problem P_0 and generated partial problems are decomposed into smaller partial problems is represented by a *finite rooted tree* $\mathcal{B}=(\mathcal{P}, \mathcal{E})$, where $\mathcal{P}$ is a set of nodes and $\mathcal{E}$ is a set of arcs. The root of $\mathcal{B}$ is denoted by P_0 and corresponds to the given problem P_0. $(P_i, P_j) \in \mathcal{E}$, i.e., P_j is a *son* of P_i, if P_j is generated from P_i by decomposition. The set of bottom (leaf) nodes $\mathcal{T}$ represents the partial problems which can be trivially solved. The *depth* of a node P_i is the length of the path from P_0 to P_i. The *height* of $\mathcal{B}$ is the maximum depth of nodes in $\mathcal{P}$, and is identified with the *size* of problem P_0.

Let $O(P_i)$ and $f(P_i)$ be the set of optimal solutions of P_i and the optimal value, respectively. O and f satisfy

$$
\begin{aligned}
f(P_i) &= \min\{f(P_{i_j}) \mid j=1, 2, \ldots, k\} \\
O(P_i) &= \cup\{O(P_{i_j}) \mid f(P_i)=f(P_{i_j}),\ j=1, 2, \ldots, k\},
\end{aligned} \tag{2.1}
$$

if P_i has sons $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$. (2.1) implies $f(P_i) \leq f(P_j)$ for a son P_j of P_i. $(\mathcal{B}, O, f)$ is called the *branching structure* of P_0. Our objective is to obtain $O(P_0)$ and $f(P_0)$, without really generating all $P_i \in \mathcal{P}$.

Note that $f(P_i)$ is usually not known but a lower bound $g(P_i)$ of $f(P_i)$ is actually calculated for each generated node P_i; g satisfies the following conditions.

(a) $g(P_i) \leq f(P_i)$ for $P_i \in \mathcal{P}$,

(b) $g(P_i) = f(P_i)$ for $P_i \in \mathcal{T}$,

(c) $g(P_i) \leq g(P_j)$ if P_j is a son of P_i.

$\mathcal{G}$ denotes the set of nodes P_i for which $f(P_i)$ is obtained or $O(P_i) \cap O(P_0) = \phi$ is concluded in the computing process of g. Then it satisfies

(A)† $g(P_i)=f(P_i)$ for $P_i \in \mathcal{G}$,

(B) $\mathcal{G} \supset \mathcal{T}$,

(C) $P_i \in \mathcal{G}$ implies $P_j \in \mathcal{G}$, if P_j is a son of P_i.

At each stage of computation, a node in $\mathcal{P}$ is called *active* if it has already been generated but neither decomposed nor terminated (i.e., solved or concluded not to yield an optimal solution of P_0) by test. Given a set of active nodes $\mathcal{A}$, a *search function* s selects a node $s(\mathcal{A})$ from $\mathcal{A}$. s is a *breadth-first search function* if $s(\mathcal{A})$ has the minimum depth among the nodes in $\mathcal{A}$. It is a *depth-first search function* if $s(\mathcal{A})$ has the maximum depth. s is called a *heuristic search function* based on h: $\mathcal{P} \to$E (reals) and is denoted by s_h, if $s_h(\mathcal{A})$ is the node in $\mathcal{A}$ with the smallest h value (e.g., [11, 14, 27, 28]). To preclude the case of tie, it is assumed that

(2.2) $\quad h(P_i) \neq h(P_j)$ for $P_i \neq P_j \in \mathcal{P}$.

If h is set equal to g, s_g is called a *best-bound search function*.

Now we present a formal description of a branch-and-bound algorithm A, which uses only the lower bound test; algorithm using the dominance test will be discussed in Section 5. It obtains all optimal solutions, i.e., $O(P_0)$, upon termination. When only a single optimal solution is sought, it can be slightly modified to improve the computational efficiency (e.g., [14]). All the results in this paper, however, can be extended with minor modification.

Branch-and-Bound Algorithm $A=((\mathcal{B}, O, f), (\mathcal{G}, g), s)$.

Remark. O denotes the set of the best feasible solutions of P_0 currently available, and is called the *incumbent*. z denotes its value. It is assumed that $O(P_i)$ is obtained as a by-product of computation of $g(P_i)$ if $P_i \in \mathcal{G}$.

A1 (Initialize)†† $\mathcal{A} \leftarrow \{P_0\}$, $z \leftarrow \infty$, $O \leftarrow \phi$(empty).

A2 (Search): If $\mathcal{A}=\phi$, go to A7; otherwise let $P_i \leftarrow s(\mathcal{A})$ and go to A3.

A3 (Test): If $P_i \in \mathcal{G}$ go to A5; otherwise go to A6 if $g(P_i)>z$, and go to A4 if $g(P_i) \leq z$.

---

† When $P_i \in \mathcal{G}$ holds because $O(P_i) \cap O(P_0)=\phi$ is concluded, $g(P_i)=f(P_i)$ may not be satisfied or $g(P_i)$ may not be computed. Even in this case, we assume for simplicity that condition (A) holds, since the value $g(P_i)$ is not relevant to the computation process.

†† $\leftarrow$ denotes the assignment operation equivalent to := in Algol.

A4 (Decompose): Generate sons $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$ of P_i. Return to A2 after letting $\mathcal{A} \leftarrow \mathcal{A} \cup \{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\} - \{P_i\}$.

A5 (Improve): Go to A6 after letting

$$O \leftarrow \begin{cases} O & \text{if } z < f(P_i)\ (=g(P_i)) \\ O \cup O(P_i) & \text{if } z = f(P_i) \\ O(P_i) & \text{if } z > f(P_i) \end{cases}$$

$$z \leftarrow \min[z, f(P_i)].$$

A6 (Terminate P_i): Return to A2 after letting $\mathcal{A} \leftarrow \mathcal{A} - \{P_i\}$.

A7 (Halt): Halt. $O(P_0) = O$ and $f(P_0) = z$; P_0 is infeasible (and $O(P_0) = \phi$) if $z = \infty$.

Note that only a small portion of $\mathcal{P}$ is actually generated in most branch-and-bound algorithms. As was noted in Section 1, the computational efficiency of A is closely related to the number of nodes actually generated, which is measured by

$T(A)$: the total number of nodes which are decomposed in A4 until the computation halts in A7.

3. A Lower Bound for T(A)

The next theorem gives a lower bound of $T(A)$ which is valid for any branch-and-bound algorithm A.

Theorem 3.1. Let $A = ((\mathcal{B}, O, f), (\mathcal{S}, g), s)$ be a branch-and-bound algorithm. Then $T(A) \geq |\mathcal{F} - \mathcal{S}|$, where $\mathcal{F} = \{P_i \in \mathcal{P} \mid g(P_i) \leq f(P_0)\}$ and $|\cdot|$ denotes the cardinality of the set therein.

Proof: Whenever a node $P_i \in \mathcal{F} - \mathcal{S}$ is tested in A3, it goes to A4 and P_i is decomposed since $P_i \notin \mathcal{S}$ and $g(P_i) \leq f(P_0) \leq z$ always holds. Furthermore each ancestor P_k of P_i satisfies $P_k \notin \mathcal{S}$ (by condition (C) of $\mathcal{S}$) and $g(P_k) \leq g(P_i)$ (by condition (c) of g) $\leq z$, and P_k is also decomposed. Thus P_i is eventually generated and tested; it is then decomposed in A4. □

4. Exponential Growth of T(A)

In this section, we analyze a probabilistic behavior of a branch-and-bound algorithm and show evidence for the exponential growth of $T(A)$ with the size of given problem P_0.

Consider a class of problems which contains a number of (possibly infinite) problems with size n for each positive integer n. It is assumed that all problems with size n has the same tree $\mathcal{B}(n)$ representing their decomposition processes. (If necessary, preclude the problems with different $\mathcal{B}(n)$.) For example, the size of an integer programming problems with n 0-1 variables is n, and an infinite number of problems with size n can be generated by specifying coefficients in the objective function and constraints. $\mathcal{B}(n)$ in this case is a $(2, n)$-tree, i.e., each node in depths 0, 1, 2,..., n-1 has two sons and all nodes in depth n are bottom nodes. (This assumes a common decomposition scheme† such that a partial problem is decomposed into two by fixing one variable to 0 and 1.)

It is also assumed that each $\mathcal{B}(n)$ has an independent set†† of nodes $\{P_i \mid i\in I(n)\}\subset \mathcal{P}-\mathcal{T}$, where $r(n)=|I(n)|$ satisfies $r(n)>c_1k_1^n$ for some $c_1, k_1>0$. This condition is satisfied in most cases encountered in practical application. In the above $(2, n)$-tree of the integer programming problem, $r(n)=(\frac{1}{2})2^n$ holds for the set of all nodes in depth n-1, and $r(n)=2^{\lfloor \frac{n}{a} \rfloor}$ holds for the set of all nodes in depth $\lfloor \frac{n}{a} \rfloor$, where a>0 is a constant.

For simplicity, we use the convention

$$f_i=f(P_i) \text{ and } g_i=g(P_i) \text{ for } i\in I(n).$$

By definition of f and g,

$$f_i\geq f_0,\ g_i\leq f_i \quad \text{for } i\in I(n) \tag{4.1}$$

---

† Generally speaking, partial problems which are generated from different integer programming problems P_0 but correspond to the same node in the branching structure $\mathcal{B}(n)$ may fix different variables to 0 and 1. Thus the partial problems corresponding to the same node are not the same in the sense that sets of fixed variables are different for different integer programming problems P_0. Even in this case, however, we can say that all problems P_0 with n variables have the same branching structure $\mathcal{B}(n)$ which is the $(2, n)$-tree, and the following discussion can include this general decomposition scheme as will be easily confirmed. For some branching structures such as those discussed in Examples 4.1 and 4.2, this consideration is unnecessary since all partial problems corresponding to the same node have exactly the same set of fixed variables.

†† $P_i, P_j\in \mathcal{P}$ are *independent* if neither is a proper descendant of the other. A set is *independent* if any two nodes therein are independent.

hold. No restriction is however a priori assumed on the relative values of f_i and f_j (or g_i and g_j) for $i,j\in I(n)$, since P_i and P_j are independent (cf. (2.1) and condition (c) of g).

When we want to investigate the behavior of a branch-and-bound algorithm for a class of problems, as a whole, it would be natural to regard f_i and g_i as random variables satisfying condition (4.1) and having a probability density function $p(f_0, f_1, f_2, \cdots, f_{r(n)}, g_1, g_2, \cdots, g_{r(n)})$ which is specific to the given class of problems. To keep the model as general as possible, we accept any probability density function provided that the marginal density function†

$$p(g_i)=\int\int\cdots\int_{-\infty}^{\infty} p(f_0,\cdots, g_{r(n)})df_0\cdots df_{r(n)}dg_1\cdots dg_{i-1}dg_{i+1}\cdots dg_{r(n)} \tag{4.2}$$

exists for each $i\in I(n)$, and furthermore

$$\int_{-\infty}^{f_0} p(g_i)dg_i\geq\varepsilon \tag{4.3}$$

holds for some constant $\varepsilon>0$ (independent of n) and for all $i\in I'(n)$, where $I'(n)\subset I(n)$ and $r'(n)=|I'(n)|$ satisfies

$$r'(n)\geq c_2 k_2^{\,n} \tag{4.4}$$

for some c_2, $k_2>0$.

Assumption (4.3) would need a justification. It says that g_i has a positive probability ε of being not greater than $f_0(=f(P_0))$ when all problems with size n are taken into account, for i in a nontrivial subset $I'(n)$ of $I(n)$. In other words, the magnitude of underestimation by lower bounding function g, i.e., f_i-g_i, can be greater than the deviation of f_i from f_0, i.e., f_i-f_0, with a positive probability ε. This may be quite natural since the class of problems under consideration would include those problems for which a partial problem P_i, $i\in I'(n)$, is difficult and the underestimation by g becomes relatively large. (See also the discussion in Examples 4.1 and 4.2.)

Now let $\overline{T}(n)$ denote the mean of $T(A)$ when a branch-and-bound algorithm A is applied to all problems with size n in the given class. By the above assumptions, $\overline{T}(n)$ satisfies

---

† f_i, g_i are assumed continuous. If f_i, g_i are discrete, the following conditions should be modified in an obvious manner.

$$\bar{T}(n) \geq \sum_{i \in I(n)} \int_{-\infty}^{f_0} p(g_i) dg_i \qquad \text{(by Theorem 3.1)}$$

$$\geq \sum_{i \in I'(n)} \varepsilon = r'(n)\varepsilon \qquad \text{(by (4.3))}$$

$$\geq (c_2 \varepsilon) k_2^n, \qquad \text{(by (4.4))}$$

proving the next theorem.

Theorem 4.1. The mean number $\bar{T}(n)$ of partial problems which are decomposed in a branch-and-bound algorithm A applied to a class of problems B, where A and B satisfy all the assumptions mentioned above, grows at least as fast as exponentially with the problem size n.

This theorem says that the exponential growth of $\bar{T}(n)$ is inevitable unless either the class of problems under consideration has a very narrow $\mathcal{S}(n)$ such that all independent sets of nodes have the cardinality less than exponential, or an exceptionally good lower bounding function g is available for which (4.3) and (4.4) do not hold. Thus we should anticipate the exponential growth of $\bar{T}(n)$ for almost all branch-and-bound algorithms using the lower bound test only, as far as a class of problems having a nontrivial complexity is concerned.

At this point, it should be noted that the implication of Theorem 4.1 is primarily theoretical since $p(g_i)$ of (4.2) is usually not known for a given particular class of problems. In some cases, however, the assumptions in Theorem 4.1 can be actually checked, as in the following examples.

Finally, we should be careful enough not to conclude from the above discussion that branch-and-bound algorithms are always useless. Even if $\bar{T}(n)$ grows exponentially, it may be possible to slow down its growth rate to a degree with which problems of meaningful sizes may be solved in practical computation time. Some examples of such practical branch-and-bound algorithms may be those currently used for the integer programming problem (e.g., [8, 9]) and the traveling salesman problem (e.g., [2, 13]), which are known to be NP-complete and hence believed to require exponential computation time by any algorithm whatever.

Example 4.1. Let N be a complete graph with n nodes 1, 2,..., n and distance matrix $[c_{ij}]$, where $c_{ij} > 0$ (possibly ∞) is the length of arc (i, j), for $1 \leq i, j \leq n$. The shortest path problem asks to find all paths from node 1 to node n having the minimum length.

Now let $\Sigma = \{1, 2, \ldots, n\}$ be the set of *decisions*, where i is the decision to make node i the next node to visit. Thus a sequence of dicisions (called

a *policy*) $x=i_1 i_2 \ldots i_k$ represents path $1 \to i_1 \to i_2 \to \ldots \to i_k$. Σ^* denotes the set of policies generated by Σ. In particular $\varepsilon \in \Sigma^*$ denotes the null policy. For each $x \in \Sigma^*$, let $P(x)$ be the partial problem to find the set of shortest paths $O(P(x))$ to node n with its initial portion restricted to be x; $f(P(x))$ denotes their length. $P(\varepsilon)(=P_0)$ stands for the original problem. Each $P(x)$ may be decomposed into n partial problems $P(x1)$, $P(x2)$,..., $P(xn)$. Thus $\mathcal{B}(n)$ is a rooted tree in which each node except for a bottom node has n sons (see Fig. 1). Bottom nodes $\mathcal{T}$ correspond to the partial problems such that either $\pi(x)=n$, where $\pi(x)$ denotes the last decision in x ($\pi(x)=\varepsilon$ if $x=\varepsilon$), or x represents a nonelementary path (i.e., x visits a node in N more than once). In the first case, x represents a path from node 1 to node n; hence $P(x)$ is trivially solved. In the latter case, $P(x)$ does not give an optimal solution by assumption $c_{ij}>0$ ($f(P(x))$ is set to ∞).

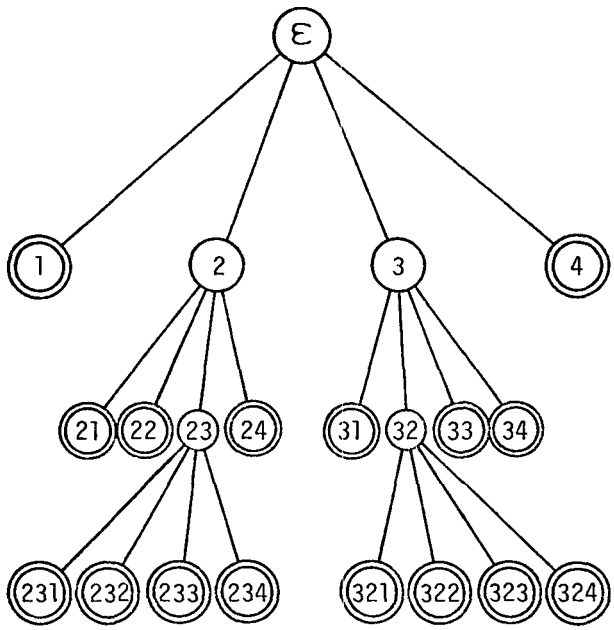

Fig. 1 Rooted tree $\mathcal{B}$ of the shortest path problem in Examples 4.1 and 5.1. (n=4) (Double circles denote bottom nodes.)

For this branching structure ($\mathcal{B}(n)$, O, f), ($\mathcal{T}$, g) may be defined as follows.

$\mathcal{G} = \mathcal{T}$

(4.5) $$g(P(x)) = \begin{cases} \infty & \text{if } P(x) \in \mathcal{T} \text{ and } x \text{ represents a nonelementary path,} \\ c_{1i_1} + c_{i_1 i_2} + \cdots + c_{i_{k-1} i_k} & \text{for } x = i_1 i_2 \cdots i_k, \text{ otherwise.} \end{cases}$$

It is not difficult to see that these $\mathcal{G}$, g satisfy all the conditions given in Section 2. Thus these give rise to a branch-and-bound algorithm $A=((\mathcal{B}(n), 0, f), (\mathcal{G}, g), s)$ for the shortest path problem.

To analyze the behavior of A, consider a special graph structure: N is the n'-stage graph with the start node in stage 0, m nodes in each of the stages 1, 2,..., $n'-1$, and the terminal node in stage n'. N has $n=mn'-m+2$ nodes. An arc exists from each node in stage i to any node in stage $i+1$, for $i=0, 1, \ldots, n'-1$. The length of each arc is determined as a random number taken independently from the same probability density function $q(x)$ defined over $[1,\infty)$. $q(x)$ can be arbitrary as far as the useal assumptions to guarantee the central limit theorem are satisfied. In particular, its mean exists and is denoted a (≥ 1). Since each arc has length not smaller than 1, the length of the shortest path in this graph is at least n'.

It is now easy to see that each partial problem, which is not a bottom node, has exactly m sons with finite g-values (of course, only nodes with finite g-values can lead to a shortest path). Thus exactly $m\lfloor \frac{n'}{a} \rfloor$ partial problems with finite g-values exist in depth $\lfloor \frac{n'}{a} \rfloor$ of the branching structure. Let $I(n)=I'(n)$ denote the set of their indices. Then $g(P_i)$ of P_i, $i \in I(n)$, is the length of a path consisting of $\frac{n'}{a}$ arcs (i.e., the sum of lengths of $\lfloor \frac{n'}{a} \rfloor$ arcs). By the central limit theorem, therefore, $g(P_i)$ is normally distributed around its mean $a\lfloor \frac{n'}{a} \rfloor$ ($\leq n'$) if n' is sufficiently large. Thus the probability that $g(P_i)$ is not greater than the length of the shortest path ($\geq n'$) is at least $\varepsilon=1/2$ (>0).

Consequently the above model satisfies all the assumptions of Theorem 4.1 and hence $\overline{T}(n)$ grows at least as fast as exponentially with n' (i.e., n). Thus a branch-and-bound approach with lower bounding function (4.5) only is not competitive with other existing algorithms (e.g., [6]) which can obtain the shortest paths in polynomial time. By incorporating a dominance relation, however, the above branch-and-bound algorithm can be improved to a polynimial time algorithm, as shown in the subsequent sections.

Example 4.2. Consider the flow-shop scheduling problem with n jobs and m machines [3]. Each job j ($1 \leq j \leq n$) is processed on machines 1, 2,..., m in this order. The processing time of job j on machine i is p_{ji} (≥ 0). When only per-

mutation schedules are permitted (i.e., all machines process jobs in the same order), find an optimal schedule (i.e., an order of n jobs on machines) which minimizes the maximum completion time (makespan) on machine m.

An obvious branching scheme for this problem is to decompose a partial problem in depth k ($\leq n$), given a subschedule for initial k jobs, into $(n-k)$ partial problems by fixing the $(k+1)$st job from the remaining $(n-k)$ jobs. In this scheme, a partial problem in depth k is decomposed into $n-k$ sons. The resulting branching structure is similar to Fig. 1 (except that no job appears more than once in a subschedule defining a partial problem). All partial problems in depth $n-1$ belong to $\mathcal{T}$ since the n-th job is then automatically determined. For partial problem P, let $J(P)$ denote the set of fixed jobs, and $t_j(P)$ denote the completion time of job j ($\in J(P)$) on machine m when jobs in $J(P)$ are processed in the order specified by P. It is then obvious that

$$g(P)=\max\{t_j(P) \mid j\in J(P)\} \tag{4.6}$$

satisfies all the conditions of a lower bounding function given in Section 2. The resulting branch-and-bound algorithm A, however, satisfies all the assumptions of Theorem 4.1 in certain cases as given below, and its $\overline{T}(n)$ must have an exponential growth.

Now assume that processing times p_{ji} are independently determined from a probability density function $q(x)$ which is defined over $[0, \infty)$ and has mean a (≥ 0). The minimum of the maximum completion time on machine m, t^m, obviously satisfies

$$t^m \geq \sum_{j=1}^{n} p_{j1} \geq \sum_{j\notin J(P)} p_{j1} \ (\equiv \bar{t}).$$

By the central limit theorem, again, $\bar{t}$ is normally distributed around its mean $a(n-|J(P)|)$ if n is sufficiently large. Next, for partial problems P with depth $\frac{n}{m+1}$ (its index set is denoted $I(n)=I'(n)$), it is obvious that

$$g(P)\leq \sum_{i=1}^{m} \sum_{j\in J(P)} p_{ji} \ (\equiv t')$$

holds, and t' is also normally distributed around its mean $ma|J(P)|=ma\frac{n}{m+1}$ by the central limit theorem. Since (1)

$$\text{mean}(\bar{t})=a(n-|J(P)|)=a(n-\tfrac{n}{m+1})\geq a\tfrac{mn}{m+1}\geq ma\tfrac{n}{m+1}=\text{mean}(t')$$

holds for $|J(P)|=\frac{n}{m+1}$, and (2) $\bar{t}$ and t' are independent random variables, the probability of $g(P)\leq t^m$ is at least $\varepsilon=1/2$. Consequently, in this case, all the assumptions of Theorem 4.1 are satisfied.

Thus a branch-and-bound algorithm with only a simple lower bounding func-

tion (4.6) would not be efficient enough to solve large problems. When m is restricted to 2, however, a powerful method is known to improve its efficiency. It will be discussed as Example 6.2.

5. Dominance Test

As mentioned in Section 1, the dominance test, which may be regarded as a generalization of the lower bound test, is sometimes incorporated in A3 (Test) of a branch-and-bound algorithm. It is based on a *dominance relation*† D which is a *partial ordering* on $\mathcal{P}$ satisfying

(i) $P_j DP_k \wedge P_j \neq P_k$ imply $f(P_j) < f(P_k)$,

(ii) $P_j DP_k \wedge P_j \neq P_k$ imply that for each descendant $P_{k'}$ of P_k, there exists a descendant $P_{j'}$ of P_j such that $P_{j'} DP_{k'}$.

By condition (i), we conclude that P_k does not provide an optimal solution of P_0 if P_j satisfying $P_j DP_k$ has already been generated. As a result, A3 may be modified as follows.

Modified A3 (Test)††: Go to A5 if $P_i \in \mathcal{S}$, and go to A6 if $g(P_i) > z$ holds or $P_j DP_i$ holds for some $P_i (\neq P_j) \in \mathcal{N}$. Otherwise go to A4. ($\mathcal{N}$ denotes the set of nodes currently generated, i.e., those which are active, terminated or decomposed.)

A branch-and-bound algorithm with this modified A3 is denoted $A=((\mathcal{B}, O, f), (\mathcal{S}, g), D, s)$.

A primal motivation of introducing a dominance relation is to improve the computational efficiency by terminating a larger number of partial problems. This point will be further discussed in the next section. It is also interesting, however, to see that almost all algorithms of combinatorial optimization problems based on dynamic programming can be viewed as branch-and-bound using a dominance relation. (This point was first noticed by Kohler [20] and

---

† This definition is for the case in which all optimal solutions of P_0 is sought. For the case of seeking a single optimal solution, it should be slightly modified [15] but the following argument also holds true without any essential change.

†† P_i is said *terminated* in A3 if one of the conditions $P_i \in \mathcal{S}$, $g(P_i) > z$ and $P_j DP_i$ holds.

the dynamic programming approach to the traveling salesman problem [12] was formulated as a branch-and-bound algorithm using a dominance relation. Morin and Marsten [26], on the other hand, attempt to improve the dynamic programming computation by using a method based on the lower bound test of branch-and-bound algorithms.) A dominance relation thus adds another dimension of flexibility both in designing efficient branch-and-bound algorithms and in unifying a wide variety of algorithms under the name of branch-and-bound.

As such an example, we consider here the well known Dijkstra algorithm for the shortest path problem which apparently has never been considered as branch-and-bound. (For the description of the Dijkstra algorithm, see [5,6].)

Example 5.1. Consider the branch-and-bound algorithm A discussed in Example 4.1 to obtain the shortest paths in a given complete graph with positive arc lengths. In addition to the lower bounding function g of (4.5), introduce a dominance relation D defined by

$$P(x)DP(y) \Leftrightarrow [\pi(x)=\pi(y) \text{ and } g(P(x))<g(P(y))]. \tag{5.1}$$

This D satisfies condition (i) since $g(P(x))<g(P(x)) \Rightarrow (\forall w\in\Sigma^*)(g(P(xw))<g(P(yw))) \Rightarrow f(x)<f(y)$. Condition (ii) is also satisfied since $P(x)DP(y) \Rightarrow (\forall w\in\Sigma^*)(P(xw)DP(yw))$, as is easily proved.

From these, we can now construct a branch-and-bound algorithm $A=((\mathcal{B}(n), O, f), (\mathcal{G}, g), D, s_g)$, where s_g is the best-bound search function mentioned in Section 2. A begins with decomposing $P(\varepsilon)$ into $P(1)$, $P(2)$,..., $P(n)$. During computation, only a partial problem $P(x)$ satisfying

$$g(P(x))\leq g(P(y)) \text{ for all } P(y)\in\mathcal{P} \text{ satisfying } \pi(y)=\pi(x) \tag{5.2}$$

can be decomposed, and the sequence of the decomposed partial problems $P(x_1)$, $P(x_2)$,..., $P(x_r)$ satisfies

$$g(P(x_1))\leq g(P(x_2))\leq \ldots \leq g(P(x_r)),$$

by virtue of D and best-bound search. As a characteristic of best-bound search, the first partial problem $P(x)$ tested in A3 satisfying $P(x)\in\mathcal{G}$ and $\pi(x)=n$ gives an optimal solution, and z is immediately set to $f(P(x))$ (=the length of the shortest path) from ∞. After then, no partial problem is decomposed in A4 (since $z\leq g(P(x'))$ for all the remaining partial problems $P(x')$), and A7 (Halt) is eventually reached.

It is now easy to see that A is essentially the same as the Dijkstra algorithm. (Although the Dijkstra algorithm is usually given in the form of obtaining a single optimal solution, the above algorithm obtains all optimal

solutions. It can be easily modified to an algorithm for obtaining a single optimal solution.)

6. Upper Bound for T(A)

An upper bound for $T(A)$ derivable from a dominance relation D is given in this section. This bound is sometimes useful to find a class of problems for which a branch-and-bound is particularly efficient, e.g., the shortest path problem of Example 5.1.

Let $A=((\mathcal{B}, O, f), (\mathcal{G}, g), D, s_h)$ be a branch-and-bound algorithm with a heuristic search function (see Section 2). Heuristic search is adopted here since it includes most of the known search strategies, such as breadth-first search, depth-first search and best-bound search, as special cases [14]. D is said *consistent with* h if $P_i DP_j$ implies $h(P_k)<h(P_j)$ for any proper ancestor P_k of P_i. By definition of heuristic search, this implies that no proper ancestor of P_i is active when P_j is selected (i.e., P_i has already been generated or a proper ancestor of P_i has been terminated). Next, D is *consistent with* g if $P_i DP_j$ implies $g(P_i)<g(P_j)$. (This notion was first introduced in [21].)

The above two types of consistency assumptions are in fact satisfied by many dominance relations proposed in the literature. For example, the dominance relation used in [16, 22] for the two-machine flow-shop scheduling problem to minimize mean flow time (instead of makespan), and the one used in [29] for the one-machine scheduling problem with deadlines are consistent with g and h, where heuristic search function s_h in these algorithms represents breadth-first search.

Lemma 6.1. D is consistent with a heuristic function h in each of the following cases.

(a)† $h(P_k)<h(P_\ell)$ if P_ℓ is a son of P_k, and $P_i DP_j \wedge P_i \neq P_j$ implies $h(P_i)<h(P_j)$.

(b) $h=g$ (i.e., best-bound search) and D is consistent with g.

(c) s_h is a breadth-first search function, and D satisfies that $P_i DP_j$ can hold only if P_i and P_j are in the same depth.

Now recall that D is a partial ordering on $\mathcal{P}$. $P_j \in \mathcal{P}$ is *minimal* with respect to D if no $P_i \neq P_j \in \mathcal{P}$ satisfies $P_i DP_j$. The set of minimal nodes is denoted by $\mathcal{P}_D$.

---

† The first portion of this assumption does not lose generality [14].

Lemma 6.2. For a branch-and-bound algorithm $A=((\mathcal{B}, O, f), (\mathcal{S}, g), D, s_h)$, assume that D is consistent with both g and h. Then any $P_j \in \mathcal{P}$ decomposed in A is minimal with respect to D.

Proof: Assume that $P_i D P_j$ and $P_i \neq P_j$. Since D is consistent with h, P_i has already been generated (i.e., $P_i \in \mathcal{N}$) or ancestor P_k of P_i has been terminated when P_j is selected. In the first case, P_j is terminated in A3 by the dominance test. Thus consider the second case; three cases are possible.

(a) P_k has been terminated in A3 by $P_k \in \mathcal{S}$. Then $z(P_j) \leq z(P_k) \leq f(P_k) \leq f(P_i)$ since P_j is selected after P_k, where $z(P_j)$ is the incumbent value when P_j is selected. Furthermore note that $f(P_i)=g(P_i)$ (since $P_i \in \mathcal{S}$ by condition (C) of $\mathcal{S}$) $< g(P_j)$ (since D is consistent with g) holds. Thus P_j is terminated in A3 by $g(P_j) > z(P_j)$.

(b) P_k has been terminated in A3 by $g(P_k) > z(P_k)$. Then $g(P_k) \leq g(P_i)$ (by condition (c) of g) $< g(P_j)$ (since D is consistent with g), and $z(P_k) \geq z(P_j)$ since P_j is selected after P_k. Thus $g(P_j) > z(P_j)$ and P_j is also terminated.

(c) P_k has been terminated in A3 by $P_a D P_k$ for some $P_a \in \mathcal{N}$. Then some descendant $P_{a'}$ of P_a satisfies $P_{a'} D P_i$ by condition (ii) of a dominance relation, and $P_{a'} D P_i \wedge P_i D P_j$ implies $P_{a'} D P_j$ since D is a partial ordering (hence transitive). Now assume that $P_{a'}$ is eventually generated in A. It is then generated before P_j by the consistency assumption with respect to h. Thus P_j is also terminated in A3. Finally consider the case in which a proper ancestor P_b of $P_{a'}$ has been terminated (and $P_{a'}$ was not generated). We may then apply the same argument as above to P_b instead of $P_{a'}$. This process, however, cannot continue indefinitely since $\mathcal{P}$ is finite. Consequently, it is proved that P_j is also terminated. □

Theorem 6.3. Let $A=((\mathcal{B}, O, f), (\mathcal{S}, g), D, s_h)$ be a branch-and-bound algorithm with D consistent with both g and h. Then $T(A) \leq |\mathcal{P}_D - \mathcal{S}|$.

Proof: Immediate from Lemma 6.2. □

The lower bound of $T(A)$ obtained in Theorem 3.1 for a branch-and-bound algorithm without using the dominance test can also be generalized to the case using it.

Theorem 6.4. Let $A=((\mathcal{B}, O, f), (\mathcal{S}, g), D, s)$ be a branch-and-bound algorithm (s may be any search function and D may be any dominance relation). Then $T(A) \geq |(\mathcal{T} - \mathcal{S}) \cap \mathcal{P}_D|$.

Proof: It was shown in the proof of Theorem 3.1 that no $P_i \in \mathcal{T} - \mathcal{S}$ can be terminated in A3 by $P_i \in \mathcal{S}$ or $g(P_i) > z(P_i)$. In addition, no $P_i \in \mathcal{P}_D$ can be terminated by $P_j D P_i$ for some $P_j \in \mathcal{N}$ since P_i is minimal with respect to D. □

Example 6.1. Consider the shortest path problem of Examples 4.1 and 5.1. By definition of D (see (5.1)), it is obvious that D is consistent with g. D is also consistent with h by Lemma 6.1 (b), since the algorithm of Example 5.1 uses best-bound search. From (4.5) and (5.1), it is also obvious that $P(x)$ is minimal with respect to D if and only if $g(P(x)) \leq g(P(y))$ for any y with $\pi(y) = \pi(x)$. Therefore, under assumption (2.2) (i.e., no two partial paths have the same length), there is exactly one minimal partial problem for each node i in the given network N. Thus $|\mathcal{P}_D| = n$, and the $P(x)$ in $\mathcal{P}_D$ with $\pi(x) = n$ satisfies $P(x) \in \mathcal{G}$. This shows that $|\mathcal{P}_D - \mathcal{G}| = n-1$ and $T(A) \geq n-1$ by Theorem 6.3. (If some partial paths have the same length, this number increases correspondingly, since all shortest paths are to be sought.)

Example 6.2. Consider the flow-shop scheduling problem introduced in Example 4.2. For this problem, no powerful dominance relation is known at present. Moreover, it is known that the flow-shop scheduling problem with $m \geq 3$ is NP-complete [7, 24], thereby suggesting that there exists no dominance relation D such that it can be tested in polynomial time whether $P_i D P_j$ holds for a given pair P_i and P_j, and $T(A)$ of the resulting branch-and-bound algorithm A is bounded by a polynomial (since otherwise A is an algorithm to solve an NP-complete problem in polynomial time, that is most unlikely).

For the flow-shop scheduling problem with $m=2$, however, there is a classical polynomial time algorithm due to Johnson [18] (see also [3]). We do not give a detailed description of the Johnson algorithm, but point out that it can be viewed as a branch-and-bound algorithm with the following dominance relation D.

(6.1) $P_k D P_\ell \Longleftrightarrow$ (a) $|J(P_k)| = |J(P_\ell)|$ (i.e., P_k and P_ℓ are in the same depth), (b) there exists exactly one† pair of jobs j_1 and j_2 such that j_1 is forced to be processed before j_2 in P_k but j_2 is forced to be processed before j_1 in P_ℓ, and (c) $\min[P_{j_1 1}, P_{j_2 2}] < \min[P_{j_2 1}, P_{j_1 2}]$.

Note that j_1 is forced to be processed before j_2 in P if either (1) $j_1, j_2 \in J(P)$ and j_1 is fixed to be processed before j_2, or (2) $j_1 \in J(P)$ and $j_2 \notin J(P)$. The proof that D of (6.1) is in fact a dominance relation is omitted since it is essentially the same as Johnson's proof [18].

Now it is not difficult to show that exactly one partial problem in each

† The restiction is necessary to make the resulting D a partial ordering.

depth of the branching structure belongs to $\mathcal{P}_D$ if $\min[P_{j_1 1}, P_{j_2 2}]=\min[P_{j_2 1}, P_{j_1 2}]$ does not hold for any pair j_1 and j_2. To apply Theorem 6.3, consider a branch-and-bound algorithm with breadth-first search function (then Lemma 6.1 (c) holds). The resulting algorithm satisfies $T(n)\leq|\mathcal{P}_D-\mathcal{G}|=n-1$. Finally it is straightforward to show that this branch-and-bound algorithm is equivalent to Johnson's original algorithm in the sense that the same sequence of partial problems are tested in both algorithms.

Acknowledgements

The author wishes to thank Professors H. Mine and T. Hasegawa of Kyoto University for their support and comments. He is also indebted to Miss T. Kanazawa for her excellent typing.

References

1. Agin, N.: Optimum Seeking with Branch and Bound. *Managenent Science*, Vol. 13 (1966), pp. B176-B185.
2. Bellmore, B. and Nemhauser, G.L.: The Traveling Salesman Problem: A Survey. *Operations Research*, Vol. 16 (1968), pp. 538-558.
3. Conway, R.W.: Maxwell, W.L., and Miller, L.W.: *Theory of Scheduling*, Addison-Wesley, Reading Mass., 1967.
4. Cook, S.A.: The Complexity of Theorem Proving Procedures. *Proc. 3rd ACM Conference on Theory of Computing*, (1970), pp. 151-158.
5. Dijkstra, E.W.: A Note on two Problems in Connexion with Graphs. *Numerische Mathematik*, vol. 1 (1959), pp. 269-271.
6. Dreyfus, S.E.: An Appraisal of Some Shortest Path Algorithms. *Operations Research*, vol. 17 (1969), pp. 394-411.
7. Garey, M.R., Johnson D.S., and Sethi, R.: The Complexity of Flowshop and Jobshop Scheduling. Technical Report No. 168, The Pennsylvania State University, Computer Science Department, 1975.
8. Geoffrion, A.M., and Marsten, R.E.: Integer Programming Algorithms: A Framework and State-of-the-Art Survey. *Management Science*, vol. 18 (1972), pp. 469-491.
9. Girfinkel, R.S., and Nemhauser, G.L.: *Integer Programming*, John Wiley and sons, New York, (1972).

10. Golomb, S.W., and Baumert, L.D.: Backtrack Programming. *Journal of the ACM*, vol. 12 (1965), pp. 516-524.
11. Hart, P.E., Nilsson, N.J., and Raphael, B.: A Formal Basis for the heuristic Determination of Minimal Cost Paths. *IEEE Transactions on System science and Cybernetics*, Vol. SSC-4 (1968), pp. 100-107.
12. Held, M., and Karp, R.M.: A Dynamic Programming Approach to Sequencing Problems. *Journal of SIAM*, vol. 10 (1962), pp. 196-210.
13. Held, M., and Karp, R.M.: The Traveling Salesman Problem and Minimum Spanning Trees : Part 2. *Mathematical Programming*, vol. 1 (1971), pp. 6-25.
14. Ibaraki, T.: Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms. *International Journal of Computer and Information sciences*, Vol. 5 (1976), pp. 315-344.
15. Ibaraki, T.: The Power of Dominance Relations in Branch-and-Bound Algorithms. Working Paper, Dept. of Applied Mathematics and Physics, Kyoto University, 1975 ; to appear in *Journal of the ACM*.
16. Ignall, E., and Schrage, L.E.: Application of the Branch and Bound Technique to Some Flow-Shop Sequencing Problems. *Operations Research*, Vol. 13 (1965), pp. 400-412.
17. Jeroslow, R.G.: Trivial Integer Programs Unsolvable by Branch-and-Bound. *Mathematical Programming*, Vol. 6 (1974), pp. 105-109.
18. Johnson, S.M.: Optimal Two- and Three-stage Production Schedules with Setup Times Included. *Naval Research Logistics Quarterly*, Vol. 1 (1954).
19. Karp, R.M.: Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, Miller, R.E., and Thatcher, J.W. (eds.), Plenum Press, New York, 1972, pp. 85-103.
20. Kohler, W.H.: Exact and Approximate Algorithms for Permutation Problems. Ph. D. Disseration, Princeton University, 1972.
21. Kohler, W.H., and Steiglitz, K.: Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems. *Journal of the ACM*, Vol. 21 (1974), 140-156.
22. Kohler, W.H., and Steiglitz, K.: Exact, Approximate, and Guaranteed Accuracy Algorithms for the Flow-Shop Problem n/2/F/$\bar{F}$. *Journal of the ACM*, Vol. 22 (1975), pp. 106-114.
23. Lawler, E.L., and Wood, D.E.: Branch-and-Bound Method : A Survey. *Operations Research*, Vol. 14 (1966), pp. 699-719.
24. Lenstra, J.K., Rinooy Kan, A.H.G., and Brucker, P.: Complexity of Machine Scheduling Problems. *Ann. Discrete Mathematics*, Vol. 1, to appear.

25. Mitten, L.G.: Branch-and-Bound Method: General Formulation and Properties. *Operations Research*, Vol. 18 (1970), pp. 24-34.
26. Morin, T.L., and Marsten, R.E.: Branch-and-bound Strategies for Dynamic Programming. *Operations Research*, Vol. 24 (1976), pp. 611-627.
27. Nilsson, N.J.: Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.
28. Pohl, I.: First Results on the Effect of Error in Heuristic Search. In *Machine Intelligence* 5, Meltzer, B., and Michie, D. (eds.), Edinburgh University Press, 1969.
29. Sahni, S.K.: Algorithms for Scheduling Independent Tasks. *Journal of the ACM*, Vol. 23 (1976), pp. 116-127.

Toshihide IBARAKI: Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University, Kyoto 606, Japan.