Journal of the Operations Research
Society of Japan
Vol. 24, No. 1, March 1981

A UNIFYING FRAMEWORK OF COMBINATORIAL OPTIMIZATION ALGORITHMS; TREE PROGRAMMING AND ITS VALIDITY

Yasuki Sekiguchi
Hokkaido University

(Received May 12, 1980; Revised August 20, 1980)

Abstract A unifying framework, named tree programming, for solution algorithms of combinatorial optimization problems, such as branch-and-bound algorithms, dynamic programming, backtrack programming, additive implicit enumeration and cutting plane methods, is proposed. Constituents of tree programming are a selection rule, a branching rule, an upper bounding function, an elimination rule and a terminating condition. The essential difference from conventional models of such algorithms is in the abstract definition of elimination rules. The validity of tree programming, finiteness and correctness, is examined. It is shown that finiteness is mainly dominated by the selection rule and the elimination rule, and that correctness by the terminating condition. As examples of tree programming, algorithms cited above are reformulated along the proposed framework.

1. Introduction

Combinatorial or discrete optimization problems have been one of the main concerns in the past two decades or more in mathematical programming research. Unfortunately, there has been no unified theory of optimal solutions but many solution algorithms have been proposed. It seems to be difficult to have a universal solution algorithm for a variety of combinatorial problems. Indeed, most of the proposed algorithms are efficient only for certain types of problems and moreover for certain types of numerical data on certain types of problems(cf. pp.16-17 in [19]). Recently, several researches on the complexity of computations have strongly suggested that the situation above comes from the nature of combinatorial problems and is not a temporary matter[11]. In addition, experience tells us that the solution algorithms for slightly different problems are usually very different. For example, an algorithm for an asymmetric travelling salesman problem is not effective for the symmetric one.

All these seem to imply that under many practical situations they must develop their own algorithms to solve their problems efficiently. Standing on

this point, possibilities of a unified theory on the solution algorithms, especially, on how to make an algorithm efficient, should be explored. This paper aims to give a basis of such research.

There are several general approaches to construct algorithms, e.g. branch-and-bound methods[1,3,9,15,17,21], dynamic programming[7,18], backtrack programming[6,23,25], cutting plane methods[5,8], etc. and many particular algorithms. But, each approach utilizes different mechanisms to make it effective. It is natural to try to compose different mechanisms into a single algorithm[9,10,13,14,18,21]. In doing this, it will be convenient to have a unified model of these algorithms.

If the scope is restricted within integer programming problems, most of the proposed algorithms are unified in the framework of Geoffrion and Marsten [5]. If restricted within the algorithms of branch-and-bound and dynamic programming types, they are unified in the framework of Kohler and Steiglitz[13] or Ibaraki[10]. Tree programming, the proposed framework, can include both types and is expected to cover majority of the conventional algorithms. Among the conventional models, Ibaraki's model seems most inclusive. But even this one is not general enough for our purpose.

Tree programming is introduced as a conceptual framework so as to help those who design or develop new solution algorithms of specific problems. For this purpose, it is not sufficient to explain representative algorithms proposed so far. Tree programming must be a help for devising new algorithms and include newly developed algorithms, too. Therefore, we will define each constituent of tree programming under as gentle restrictions as possible, and sufficient conditions for its validity will be made clear. This approach of analysis is essentially different from the conventional one, where an abstract model of representative algorithms is constructed and its validity is proved. The sufficient conditions given by the analysis tell us what must be shown so as to ensure validity of an invented algorithm, but they do not tell whether a specific algorithm is valid or not.

In section 2, several basic concepts are introduced. The underlying idea is that selecting an algorithm as a solution method causes selection of one formulation of a raw, real world problem and that different solution algorithms for a problem solve different problems in a mathematical sense.

A general framework of algorithms, i.e. tree programming, is proposed in section 3. Most algorithms of combinatorial problems are of enumerative or itrative property. Their key steps can be recognized as (1) decomposing problems into subproblems easier to solve, and (2) eliminating some of the generated subproblems which have no better solutions than the one already known.

Tree programming is a general model of this type of algorithms. It is not a specific algorithm. But it can reduce to a variety of specific algorithms when its constituents are specified.

The conditions under which tree programming is valid are discussed in section 4. The validity here includes finiteness and correctness(i.e. the accuracyof the solutions obtained by tree progamming). Similar conditions but under restricted situations are discussed in Mitten[17], Kohler and Steiglitz[13,14] or Balas[3].

In section 5, four representative algorithms; branch-and-bound methods, dynamic programming, the additive implicit enumeration and cutting plane methods are reformulated under the framework of tree programming. This section aims to illustrate the unifying capability of tree programming.

2. Problem description and enumeration

In this section, we will clarify elemental functions which tree programming must contain, through analysis of enumerative algorithms. The principal form of enumerative algorithms is seen in complete enumeration, which is the direct concern here. But, enumerative algorithms in general are concerned in the succeeding sections. Enumerative algorithms in general is not a strict expression. Tree programming is actually a framework for algorithms with a structure similar to enumerative algorithms. For example, the domain of search defined below is allowed to be an infinite set. The most important for enumerative algorithms is that an original problem is reduced to a finite number of solvable problems within a finite computation.

A solution algorithm is designed for a type of problems. Define Ω as the set of admissible data ω for a solution algorithm. Let f and F be the objective function and the set of feasible solutions defined by ω. Strictly, notations such as $f(\omega)$ and $F(\omega)$ should be used. But all variables concerning (sub)problems in this paper depend on ω. Therefore, suppressing ω from these variables will make descriptions simple without confusion. An original problem to be solved can be represented as follows.

(2.1) "Find $x^* \in F$ such that $f(x^*)=\min_{x \in F} f(x)$"

In solving (2.1), when no explicit function from F to X^*(X^* is the set of all optimal solutions in F), enumerative methods are often adopted and one or more optimal solutions are explored. In such cases, it frequently occurs that the method explores only a part of F or beyond F. Let X be the domain of search assumed by the method. The problem actually solved by the method is

(2.2) [P_o] "Find $x^* \in F \cap X$ such that $f(x^*)=\min_{x \in F} f(x)$"

The concept of the domain of search is a generalization of Agin's implicit constraints[1], and introduced for the purpose of recognizing explicitly the fact noted above. (2.2) is for single optimal solution cases. The readers will feel no difficulty in making a similar definition for all optimal solution cases. We assume in this paper the following two conditions are satisfied.

Al_a: $X^* \subset X$ in all optimal solution cases.

Al_s: $X^* \cap X \neq \phi$ if $X^* \neq \phi$ in single optimal solution cases.

For simplicity, we denote the minimum value of f over $X' \cap F (X' \subset X)$ by $f(X')$;

(2.3) $$f(X')=\begin{cases} \min_{x \in X' \cap F} f(x) & \text{if } X' \cap F \neq \phi \\ +\infty & \text{otherwise.} \end{cases}$$

The domain of search of dynamic programming is usually larger than F and that of simplex methods of linear programming is smaller than F(i.e. only vertices of a feasible region are estimated). But, a better example would be the next.

Example 1. 0-1 all integer liner programming

Find $x^* \in F$ such that $f(x^*)=\min_{x \in F} cx$

Here, $F=\{x \mid Ax \geqq b,\ x_i=0 \text{ or } 1,\ i=1,2,\ldots,n\}$, $x=(x_1,x_2,\ldots,x_n)^t$, t implies transpose. When Balas's implicit enumeration algorithm is adopted,

$$X=\{x \mid x_i=0 \text{ or } 1,\ i=1,2,\ldots,n\}.$$

We see that $F \subset X$. If one of the cutting plane methods is adopted, we can understand that

$$X=\{x \mid Ax \geqq b,\ cx \leqq f(x^*),\ 0 \leqq x_i \leqq 1,\ i=1,2,\ldots,n\}.$$

Notice that $F \cap X=X^*$, and that X in this case is distinctively different from X in Balas's algorithm. Notice also that X in cutting plane methods varies from ω to ω. In Balas's algorithm X is n-cube and never be affected by ω.

Assume the existence of a mechanism which replaces a problem with a set of subproblems and which can be implemented repeatedly. Problems generated by the mechanism are assumed to be described by ω and additional information γ given by the mechanism. $P(\gamma)$ denotes such a problem of P_o. Let $X(P)$ be the domain of search of $P(\gamma)$(the symbol P is often used for $P(\gamma)$). $X(P)$ is a subset of X. Definitely, $P(\gamma)$ is the following.

(2.4) [$P(\gamma)$] "Find $\bar{x} \in X(P) \cap F$ such that $f(\bar{x})=f(X(P))$, under γ."

$X(P)$ is determined by ω and γ, but γ may contain other additional information useful for solving P(cf. remark 2 on example 2). X is usually used for $X(P_o)$.

Let Π be the set of P_o and all subproblems which can be generated by implementing repeatedly a replacing mechanism on P_o. Assume $P \in \Pi$ is replaced

with a set of subproblems $BR(P)$. Then, it is possible to construct rooted trees $t=(N,E)$, where $N\subset\Pi$ and $E\subset N\times N$, such that

T1 : the root is $P_0(=P(\phi))$,

T2 : $(P,P')\in E$ if and only if $P'\in BR(P)$, and

T3 : $BR(P)\subset N$ if $BR(P)\cap N\neq\phi$.

Replacing mechanisms which permit construction of this type of trees are called branching rules and additional data γ attached through a branching process(i.e. implementing a replacing mechanism) is called a branching history, hereafter. A branching rule generates subproblems $BR(P)$ by adding further restrictions on P. Therefore, if $P(\gamma)$ is generated by implementing a branching rule n-times on P_0, γ is a sequence of information added by branching. As properties of replacing mechanisms, we assume the next three. For sequences, A and B, $A\subset B$ implies that A is a subsequence of B. The set of A's constituents is denoted by $\{A\}$.

¶1 : $\gamma\subsetneq\gamma'$ if and only if $P'(\gamma')$ is a subproblem of $P(\gamma)$.

¶2 : $X(P')\subset X(P)$ if $\gamma\subset\gamma'$ for $P'(\gamma')$ and $P(\gamma)$.

¶3 : $\cup\{X(P') \mid P'\in BR(P)\} = X(P)$.

A branching rule has to add nonempty information(¶1), the information usually restricts the domain of search(¶2) and no part of the domain of search $X(P)$ can be cut off(¶3).

In an algorithm, only one P can be replaced by its subproblems at a time. A rule for determining a problem on which the branching rule is implemented next is necessary in order to generate a tree t systematically. Let us call such a tree a selection rule. If a selection rule is given, a sequence of trees can be defined corresponding to a sequence of selected problems(Fig. 1).

Let $L(t)$ be the set of leaf nodes in t and $Z\subset\Pi$ be the set of all subproblems, which can be solved with greatly less computational requirement than P_0 or which are recognized as ones whose solutions are trivial. Assume BR can replace P_0 by a desired set T of subproblems in finite iterations. T is called a terminating condition. In other words, Z is a set of subproblems which an algorithm designer determines should be solved without replacing it into finer subproblems, and T shows the satisfactory degree of fineness of search. A proper combination of a branching rule, a selection rule and a terminating condition can generate a finite sequence of rooted trees $t=(t_0,t_1,t_2,\ldots,t_n)$ such that

T4 : $t_0=(\{P_0\},\phi)$ and $t_i(i=1,2,\ldots,n)$ satisfy T1 through T3,

T5 : $|L(t_i) - \{L(t_i)\cap L(t_{i+1})\}|=1$ ($|\cdot|$ is the cardinarity of a set),

T6 : $L(t_n)\subset T$.

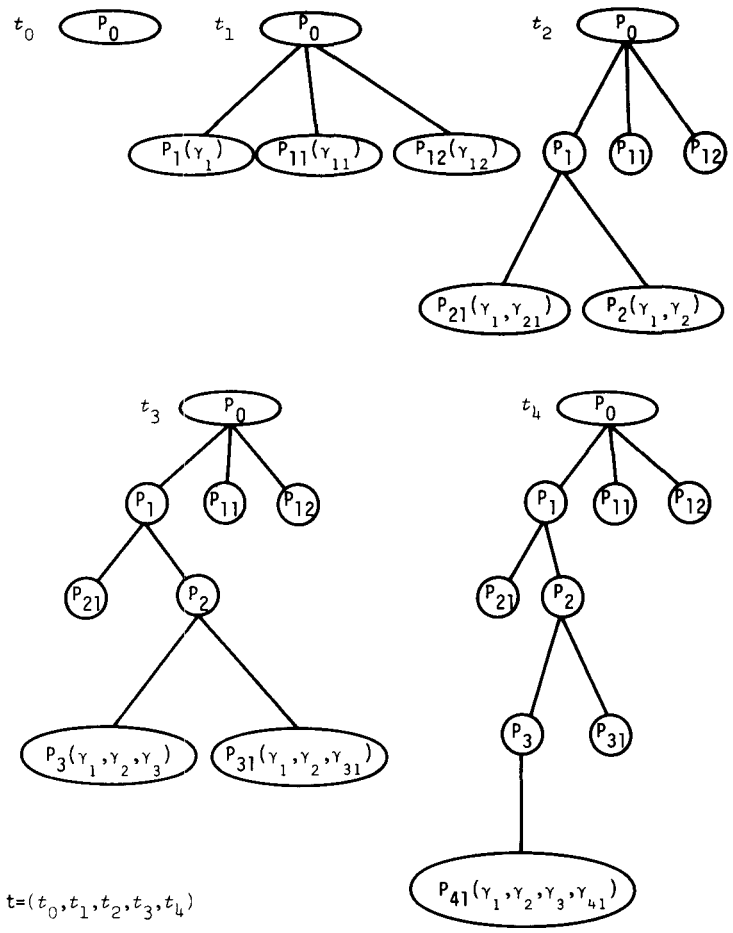

Fig. 1. A sequence of trees corresponding to a sequence of selected problems P_0, P_1, P_2, P_3, P_4 and $T=\{P_{11},P_{12},P_{21},P_{31},P_{41}\}$.

We call each t_i in t a search tree and t a search tree sequence(STS). Notice that the problem in the set $L(t_i) - \{L(t_i) \cap L(t_{i+1})\}$ is the i-th branched-from node. T6 represents the state of a search at termination. Remember that elimination, i.e. discarding some of generated subproblems from further consideration, has not been referred to, yet. T6 shows a special case of termination.

Complete enumeration algorithms can be reconstructed as a combination of a selection rule, a branching rule and a terminating condition, where $T=Z$. They generate STS's systematically. Many implicit enumeration algorithms can also be recognized as improved versions of such combinations. Before going through detailed discussions of such algorithms, an example will confirm the concepts introduced up to now.

Example 2. Active feasible schedule enumeration in job-shop

An algorithm generating all active feasible schedules for a static job-shop scheduling problem with v machines and w jobs, presented in Baker[2], is restated along the line stated above. Notations needed for describing the algorithm are defined first(see for detail[2]). $O=\{1,2,\ldots,n\}$ is the set of operations to be scheduled and $\gamma_k=\{(i_1,\tau_1),(i_2,\tau_2),\ldots,(i_k,\tau_k)\}$ is a partial active schedule, where $\{i_1,i_2,\ldots,i_k\}$ is a set of operations already scheduled and τ_j is the completion time of operation i_j. $\bar{O}_k$ is the set of operations not contained in γ_k; $\bar{O}_k=O-\{i_1,\ldots,i_k\}$. P_k is a problem of enumerating all active feasible schedules which are made by concatinating schedules of $\bar{O}_k$ to γ_k, i.e. completions of γ_k. R_k is the set of operations in $\bar{O}_k$ whose predecessors are all in γ_k. Let ψ_j be the earliest completion time and σ_j the earliest starting time of operation $j\in\bar{O}_k$ under γ_k. $m(j)$ and $r(j)$ are the machine and the processing time required by operation j, respectively. In the followings, $\leftarrow$ and $\rightarrow$ imply substitution and implication, respectively.

(algorithm)

Step 1 : (preparation) $N_0=L(t_0)\leftarrow\{P_0\}$, $E_0=\gamma_0\leftarrow\phi$, $i\leftarrow 0$.

Step 2 : (selection) Select the first generated subproblem in $L(t_i)$. Let P_k be the selected one.

Step 3 : Let $\psi^*=\min\{\psi_q \mid q\in R_k\}$ and $m^*=m(j^*)$, where $j^*\in\{j\in R_k \mid \psi_j=\psi^*\}$. Define Σ as $\{j\in R_k \mid \sigma_j<\psi^*,\ m(j)=m^*\}$.

(branching) $BR(P_k)\leftarrow\{P_{k+1} \mid \gamma_{k+1}=\gamma_k\cup(j,\sigma_j+r(j)),\ j\in\Sigma\}$.

(growing) $N_{i+1}\leftarrow N_i\cup BR(P_k)$, $E_{i+1}\leftarrow E_i\cup\{(P_k,P_{k+1}) \mid P_{k+1}\in BR(P_k)\}$, $L(t_{i+1})\leftarrow(L(t_i)-\{P_k\})\cup BR(P_k)$, where $t_{i+1}=(N_{i+1},E_{i+1})$.

Step 4 : (termination) If $k=n$ for all P_k's in $L(t_{i+1})$, terminate. Otherwise, go to step 2 after $i\leftarrow i+1$.

Remarks on example 2

1) X for the problem is the set of all active feasible schedules and $X(P_k)$ is the completion set of γ_k.

2) A partial schedule γ_k is a branching history. If $|\Sigma|=1$ for a P_k, $|BR(P_k)|=1$ and $X(P_k)=X(P_{k+1})$. But notice that $\gamma_k\subsetneq\gamma_{k+1}$. This implies P_{k+1} is solved more easily than P_k, because $|\bar{O}_{k+1}|<|\bar{O}_k|$

3) Step 2 is a selection rule and (branching) in step 3 is a branching rule. Let $T=Z$, where Z is the set of all P_n(the solution of P_n is trivial). Then,

Step 4 can be represented by using a condition $L(t_{i+1})\subset T$.

4) It will be easily understood that $t=(t_0,t_1,t_2,\ldots\ldots)$ is STS.

3. Tree programming formulation

This section gives a formal description of tree programming. The name stands for the fact that the algorithm generates STS systematically. The validity of tree programming is discussed in the succeeding section.

Recall that a branching rule is a mechanism which replaces a problem with a set of subproblems with additional information γ, i.e. a branching history. We start by defining such information. Let γ be (a sequence of) information which restricts the domain of search to be considered further within a subset of X, or which is useful in solving P_0. $P(\gamma)$ is defined as before(cf. (2.4)). $\Gamma(\phi\varepsilon\Gamma)$ is a set of γ's. Γ may include infinite-length sequences. $\Gamma^*\subset\Gamma$ denotes the set of maximal sequences among finite-length ones, i.e.

(3.1) $\Gamma^*=\{\gamma\varepsilon\Gamma \mid |\{\gamma\}|<+\infty,\ \gamma\not\subset\gamma' \text{ for any } \gamma'\varepsilon\Gamma-\{\gamma\}\}$.

Corresponding to Γ and Γ^* , define Π and Z as the sets of subproblems emerged by $\gamma\varepsilon\Gamma$ and $\gamma\varepsilon\Gamma^*$, respectively, i.e.

(3.2) $\Pi=\{P(\gamma) \mid \gamma\varepsilon\Gamma\}$, $Z=\{P(\gamma) \mid \gamma\varepsilon\Gamma^*\}$.

Let Λ be the class of independent sets in Π($A\subset\Pi$ is said to be an independent set in Π if $\gamma\not\subset\gamma'$ and $\gamma'\not\subset\gamma$ hold true for any pair, $P(\gamma)$ and $P'(\gamma')$, of elements in A).

Π(i.e. Γ) is assumed to satisfy two conditions below.

(a)Z is a set of trivial problems. $P\varepsilon\Pi$ is said to be trivial if (the designer of an algorithm judges that) P can be solved by a proper procedure and does not need to be decomposed into finer subproblems.

(b)A mapping $BR : \Pi-Z \longrightarrow \Lambda$ exists and satisfies B1 through B4($P_i=P(\gamma_i)$).

B1 : $\cup\{X(P) \mid P\varepsilon BR(P_i)\}=X(P_i)$.

B2 : $(\forall P\varepsilon\Pi-\{P_0\})\ (\exists P'\varepsilon\Pi-Z)\ (P\varepsilon BR(P'))$.

B3 : $(\forall P(\gamma)\varepsilon BR(P_i))\ (\gamma_i\subsetneq\gamma)$.

B4 : $(\forall P\varepsilon\Pi-Z)\ (|BR(P)|<+\infty)$.

BR is an abstract expression of branching rules in the previous section. B1 requires that any part of the domain of search never be cut off by BR, moreover it implies the satisfication of ¶2 and ¶3. $X(P)$'s are not needed to disjoint each other. But, it is known that BR's which disjoint $X(P)$'s usually sharpens tree programming(see Balas[3]). Notice also that $X(P)=X(P')$ can happen for $P'\varepsilon BR(P)$. B3 requires that the generated subproblems must have more information than their parent($P'\varepsilon BR(P)$ is called a son of P and P is said to

be a parent of P'). The definition of Λ, with B3, ensures ¶1. The readers will easily understand that the elements in Π can be arranged in a tree where a branch from $P(\gamma)$ to $P'(\gamma')$ corresponds to the relation $P'\in BR(P)$. This tree satisfies T1 through T3. Denote as t a subtree of this one satisfying the same conditions. Let Ξ be the class of t's(including the original one).

A selection rule is defined as a mapping $SR : \Lambda \rightarrow \Pi\text{-}Z$ such that

S1 : $(\forall A\in\Lambda)\ (SR(A)\in A)$, and

S2 : $(\forall A,A'\in\Lambda)\ (SR(A')\in A\subset A' \rightarrow SR(A)=SR(A'))$.

That the range of SR is $\Pi\text{-}Z$ corresponds to T5, and S2 is a consistency condition of SR, i.e. if a problem in $A(\subset A')$ is selected among problems in the larger set A', then the same problem must be selected among problems in A(cf. Fig. 2).

A terminating condition T can be defined as a subset of Π, i.e. $T\subset\Pi$, as in the previous section. If a combination of SR, BR and T is proper, a STS satisfying T4 through T6 will be given by starting with $t_0=(\{P_0\},\phi)$ and by repeating SR and BR alternatively(cf. remark 6 on the definition of tree programming).

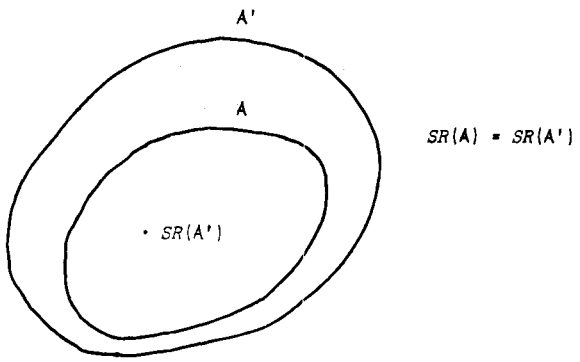

Fig. 2. A schematic representation of S2.

Most enumerative algorithms have systematic fathoming devices. Fathoming is to examine generated subproblems and to explore subproblems proved not to have valuable feasible solutions. A subproblem is said to be fathomed if it is proved not to have valuable feasible solutions. Fathomed subproblems can be excluded from further consideration. Mechanisms for fathoming and excluding subproblems are called elimination rules. An elimination rule is a mapping $ER : \Xi\times\Lambda \rightarrow \Lambda$ such that

E1 : $(\forall t=(N,E)\in\Xi)\ (\forall A\subset L(t))\ (ER(t,A)\subset A)$,

$E2_a$: $X^*\subset U\{X(P) \mid P\in A\} \rightarrow X^*\subset U\{X(P) \mid P\in ER(t,A)\}$ and

$E2_s$: $X^*\cap U\{X(P) \mid P\in A\}\neq\phi \rightarrow X^*\cap U\{X(P) \mid P\in ER(t,A)\}\neq\phi$.

$E2_a$ requires that any optimal solutions never be lost in all optimal solution cases. $E2_s$ requires that at least one optimal solution must be in the remainder $ER(t,A)$ in single optimal solution cases. For purposes of convenience, ER is said to be an identity mapping if $ER(t,A)=A$ holds true for any $t=(N,E)\in\Xi$ and $A\subset L(t)$.

The last constituent to be defined is a device to preserve the best solution found out so far. For this purpose, we define an upper bounding function as a mapping u: $\Pi \longrightarrow E$ ($E=\{\text{reals}\}\cup\{+\infty\}$) such that

$$u(P)=\begin{cases} f(X(P)) & \text{if } P\in Z \\ f(x) \text{ for some known } x\in X(P)\cap F, \text{ or } +\infty & \text{otherwise.}\end{cases} \tag{3.3}$$

The definition in (3.3) is stronger than one for an upper bounding function u_g of a lower bound test(cf. section 5) and u is usually used as u_g.

Each definition above is simply a formal description of a elemental function commonly necessary for enumerative algorithms. They are not abstraction of constituents of conventional algorithms. This is a distinctive feature of our approach, compared to the preceding researches[1,3,5,6,9,10,12-14 etc.], where abstract models of conventional algorithms were developed. Further specifications of each constituent will be given as sufficient or necessary conditions for tree programming having some properties(e.g. [22]).

It is now possible to define tree programming in a general form. The indexes, s and a, designate single optimal and all optimal solution cases, respectively. X and ER must satisfy A1 and E2 with the respective index. I denotes the best solution(s) obtained so far, i.e. incumbent(s). z denotes the value of the objective function for $x\in I$. A_i is the set of subproblems in L_i, which are not eliminated when the i-th selection is implemented, i.e. active node set. If the initial incumbent x_0 is not known at step 1, let $f(x_0)=+\infty$. As before put $t_i=(N_i,E_i)$ and $t=(t_0,t_1,t_2,\cdots)$. Noticing the fact that the effect of elimination rules is only to prune search trees, t of a tree programming TP can be said a STS if TP is finite. The name tree programming stands for this fact. The sequence of active node sets (SAN) is defined as $A(TP)=(A_0,A_1,\cdots)$. The sequence of selected nodes (SSN) is defined as $S(TP)=(P_0,P_1,P_2,\cdots)$. Notice that SAN and SSN are uniquely determined by t, and vice versa. Sometimes, the set of subproblems which are surely eliminated in TP is implicitly known. This set is referred as $\mathcal{D}$-set of TP, and is denoted as $\mathcal{D}(TP)$. $\mathcal{D}(TP)$ is assumed to include all descendants of P if $P\in\mathcal{D}(TP)$. Consider a branch-and-bound algorithm in which SR is the best bound selection type. Subproblems with lower bounds greater than $f(X(P_0))$ are surely eliminated. Then, $\mathcal{D}(TP)$ for this algorithm is equal to $\{P\in\Pi \mid g(P)>f(X(P_0))\}$, see section 5 for g. $\mathcal{D}(TP)$ of a dynamic programming algorithm with the breadth-first selection can be identi-

fied, too. General comments on the definition below will be given afterword.

[$TP_s=(SR,BR,u,ER_s,T)$] single optimal solution cases

Step 1 : (preparation) $N_0=L_0=A_0 \leftarrow \{P_0\}$, $E_0 \leftarrow \phi$, $I \leftarrow \tilde{x}_0$, $z \leftarrow f(\tilde{x}_0)$, $i \leftarrow 0$.

Step 2 : (selection) $P_i \leftarrow SR(A_i)$.

Step 3 i) : (branching) $\mathcal{B}_i \leftarrow BR(P_i)$.

ii) : (incumbent) If $\tilde{f}<z$, then $z \leftarrow \tilde{f}$ and $I \leftarrow \tilde{x}$.

Here, $\tilde{f}=\min\{u(P) \mid P\in\mathcal{B}_i\}$, and $\tilde{x}\in X(P)\cap F$ such that $P\in\mathcal{B}_i$ and $f(\tilde{x})=\tilde{f}$.

iii) : (growing) $N_{i+1} \leftarrow N_i \cup \mathcal{B}_i$, $L_{i+1} \leftarrow (L_i-\{P_i\}) \cup \mathcal{B}_i$,

$E_{i+1} \leftarrow E_i \cup \{(P_i,P) \mid P\in\mathcal{B}_i\}$, and let $t_{i+1}=(N_{i+1},E_{i+1})$.

Step 4 : (elimination) $A_{i+1} \leftarrow ER_s(t_{i+1},(A_i-\{P_i\}) \cup \mathcal{B}_i)$.

Step 5 : (termination) If $A_{i+1}\subset T$, terminate. Otherwise go to step 2 after $i \leftarrow i+1$.

In many practical situations, it is sufficient to get a single optimal solution. But, if all optimal solutions are wanted, use TP_a for TP_s.

[$TP_a=(SR,BR,u,ER_a,T)$] all optimal solution cases

Step 1,2 and 3 i) are the same as in TP_s.

Step 3 ii) : (incumbent) If $\tilde{f}<z$, then $z \leftarrow \tilde{f}$ and $I \leftarrow \tilde{X}$.

If $\tilde{f}=z$, then $I \leftarrow I\cup\tilde{X}$.

Here, $\tilde{f}=\min\{u(P) \mid P\in\mathcal{B}_i\}$, and $\tilde{X}=\{\tilde{x}\in X(P)\cap F \mid P\in\mathcal{B}_i,\ f(\tilde{x})=\tilde{f}\}$.

Step 3 iii), 4 and 5 are the same as in TP_s.

Remarks on the definitions

1) There are many researches or surveyes(cf. references) on kinds of enumerative algorithms and various formalisms are proposed. The definitions here is similar to that of Kohler and Steiglitz[13,14], Geoffrion and Marsten[5] and Baker[2] in the sense that eliminations are done right after branchings. In another class of definitions[Ibaraki 9,10], eliminations are done right after selections and the candidate to be eliminated is only the problem just selected. Relations among these formulations are discussed in [24].

2) In a concrete algorithm, it may be trivial from the property of the branching rule that some of the problems in $\mathcal{B}_i$ (e.g. see example 5 and 6 in section 5) do not need further considerations, or that they are infeasible. In such cases, the whole or a part of *ER* becomes a formality. But, problems to be eliminated are rarely known previously and *ER* is usually given as a procedure for testing each active problems. Results of the test depend on a search tree at the time of the test implementation. For example, a lower bound test can not be successful before the first feasible solution is obtained. Therefore, *ER* was

defined as one that depends on t as well as A.

3) There are many cases where memory space and computation time are restricted. In order to treat such cases, define terminating conditions as 3-tuple (T,M,C) where M is the maximal allowable memory space and C is the maximal allowable computation time(see [14] for detail).

4) Each operation is definitely separated from the others in the definitions. Practically, it sometimes happens that the amount of computation distinguishably increases if all computation concerning to an operation is gathered at its position in the definitions, or that the computation at step 3 ii) is almost the same as the one in step 4. The main purpose of the general definitions is to obtain systematic aspects over various enumerative algorithms. To give universal guides for getting an efficient computer program of a solution algorithm for a certain problem is out of the scope of the definitions. In the author's opinion, before going on to designing a program of tree programming, the combination of its constituents should be considered by referring to general properties of tree programming derived under general definitions(see [9,10,13,14,22] as examples of such properties). The similar idea is given in IV of [5].

5) A morphological scheme of various implicit enumeration algorithms is given in Müller-Merbach[19], where conventional algorithms are classified from fifteen different view points. Each of those fifteen classifications can be recognized as a classification of one of five constituents in TP.

6) Notice that if *ER* is an identical mapping in TP and if $T=Z$, the TP is a complete enumeration algorithm(compare TP to example 2). Complete enumeration algorithms here terminate when all leaf nodes are active and in Z. They are different from usual ones that terminate when all leaf nodes are eliminated. In this paper, such algorithms are recognized as implicit enumerations with a primitive lower bound test. This kind of TP_s terminates when one subproblem is active and included in Z, even if $T=Z$. This subproblem gives an optimal solution in I, and in actual places it can be eliminated also. But this must not be done under the definition of elimination rules here. One of merits of our definition is explained in remark 6 of chapter 4.

7) Subproblems in $\mathcal{D}(TP)$ as well as ones in Z are never selected. Z can be understood as an expression of algorithm designer's judge on which problems are directly solvable. $\mathcal{D}(TP)$ is a proved effect of a constituent combination, especially of *SR* and *ER*. This is the reason why they are distinguished. On the other hand, a subproblem in T may be selected. For example, in a guaranteed accuracy algorithm(cf. example 3 in chapter 5) with a depth-first selection, P such that $g(P)>(1-\xi)z$ can be selected. This is an essential difference from

the guaranteed accuracy algorithm in [12], where approximation is done through a modification of the lower bound test. If T is fixed to Z, approximations by the terminating condition can not be explained. Notice that the effect of approximations by T on the complexity of computation is different from that of approximations by the lower bound test. Notice also that subproblems in $\mathcal{D}(TP)$ are not solvable and can not be included in Z.

8) Readers may claim that $\mathcal{D}(TP)$ of the branch-and-bound algorithm with the best bound selection can be included in T, i.e. $T=Z\cup\mathcal{D}(TP)$. If this was done, no subproblem is eliminated by the test. Thus $\mathcal{D}(TP)$ must be empty by the definition, and we must understand that T is put equal to $Z\cup T_1$, where $T_1=\{P\in\Pi \mid g(P)>f(X(P_0))\}$. This T corresponds to the case (4.6a) or (4.6c), see remark 5 in section 4.

4. Validity of tree programming

Here are discussed

(1) under what conditions finiteness of tree programming is proved, and

(2) under what conditions tree programming terminates with optimal solutions.

First, the former is discussed.

Let $N(TP)$ be the number of selected nodes at step 2 in TP for a $\omega\in\Omega$ until the TP terminates, i.e. the length of SSN. TP is said finite if $N(TP)$ is finite for any $\omega\in\Omega$.

Theorem 1. Let $TP=(SR,BR,u,ER:\text{identity},T)$, $TP'=(SR,BR,u,ER',T)$ and $TP''=(SR,BR,u,ER',T')$. Assume $T\subset T'$. Then, (4.1) and (4.2) hold true. (The superfixes below imply the correspondence to TP's with the same superfixes.)

$$(4.1)\qquad (\forall P'_i\in S(TP'))\ (\exists P_{j(i)}\in S(TP))$$
$$((P'_i=P_{j(i)})\wedge(i\leq j(i)<j(i+1))\wedge(A'_i\subset A_k,\ k=j(i-1)+1,\ldots,j(i)))$$

$$(4.2)\qquad (\forall P''_i\in S(TP''))\ ((P''_i=P'_i)\wedge(A''_i=A'_i))$$

Proof: (by induction) Notice that $P'_0=P_0$ because $A'_0=A_0=\{P_0\}$, and therefore $A'_1\subset A_1$. Assume (4.1) holds for $i=l-1(0<l)$. If $P'_l=P_{j(l-1)+1}$, the proof of (4.1) is completed. So assume $P'_l\neq P_{j(l-1)+1}$. Notice that $A'_{l-1}\subset A_{j(l-1)}$ and that ER is an identity mapping. Necessarily, $A'_l\subset A_{j(l-1)+1}$. Therefore, if TP is finite, $P_{j(l-1)+\alpha}(=P'_l)$ must be in $S(TP)$. Let $P_{j(l-1)+\sigma}(\sigma<\alpha)$ be the first element in $S(TP)$ such that $P_{j(l-1)+\sigma}\in A'_l$. Then, both $P_{j(l-1)+\sigma}$ and $P_{j(l-1)+\alpha}$ are included in $A'_l\subset A_{j(l-1)+\sigma}$, and $SR(A'_l)\neq SR(A_{j(l-1)+\sigma})$. This contradicts to S2. This implies that $A'_l\subset A_{j(l-1)+\mu}(1\leq\mu\leq\alpha)$ must hold. As a result, (4.1) holds

for $i=l$. This completes the induction for (4.1).

Notice that $P''_0=P'_0=P_o$ and $A''_0=A'_0=\{P_o\}$. Assume (4.2) is true for $i=l$, then we see that

$$BR(SR(A''_l))=BR(SR(A'_l)) \qquad \text{(step 3 i)),}$$

$$A''_{l+1}=ER(t_{l+1},A)=A'_{l+1} \qquad \text{(step 4)} \qquad \text{and}$$

$$SR(A''_{l+1})=SR(A'_{l+1}) \qquad \text{(step 2).}$$

Here, $A=(A''_l-\{P''_l\})\cup BR(P'')=(A'_l-\{P'_l\})\cup BR(P'_l)$. Hence, $P''_{l+1}=P'_{l+1}$ must hold. (4.2) is established by induction. This terminates our proof.

Corollary 1. $N(TP')<+\infty \rightarrow N(TP'') \leq N(TP') \leq N(TP)$

Proof: Immediate from theorem 1.

Finiteness of TP is established if it is shown that SR emerges a finite sequence of selected nodes, which leads to an active node set included in T. Leaf nodes not included in T at the termination must have been eliminated. But notice that a subproblem is guaranteed to be eliminated only when it is included in $\mathcal{D}(TP)$. Conversely, subproblems not included in $\mathcal{D}(TP)$ may or may not be eliminated. Thus, the effect of ER on finiteness is condensed into $\mathcal{D}(TP)$. Moreover, subproblems are generated by BR. It is natural that the conditions for finiteness are expressed in terms of SR, BR, T and $\mathcal{D}(TP)$.

Theorem 2. If conditions 1, 2 and 3 hold true for any $\omega\in\Omega$, then TP is finite.

[cond. 1] $(\forall P\in T-Z)\ (BR(P)\subset T\cup\mathcal{D}(TP))$.

[cond. 2] $(SR(A_i)\in T-Z) \rightarrow (\exists n<+\infty)(SR(A_{i+n})\notin T-Z)$.

[cond. 3] $(\exists m<+\infty)\ (BR^m(P_0)\subset T\cup\mathcal{D}(TP))$, where

$$BR^0(P)=P$$
$$BR^{i+1}(P)=\{P'\in BR(P'') \mid P''\in BR^i(P)-Z\cup\mathcal{D}(TP)\}\cup\{BR^i(P)\cap(Z\cup\mathcal{D}(TP))\}. \tag{4.3}$$

Proof: Let $A=A_i-T\cup\mathcal{D}(TP)$ $(\neq\phi)$ for $i<+\infty$. By condition 3, any $P\in A$ must fall into $T\cup\mathcal{D}(TP)$ in finite iterations of BR. Because if $P\in T-Z$, all descendants of P must be included in $T\cup\mathcal{D}(TP)$ by condition 1, and because $|A|<+\infty$ by B4, the proof is completed if it is shown that $P\in A$ and its descendants included in $\Pi-T\cup\mathcal{D}(TP)$ are selected within finite iterations of step 2. Notice that

$$SR(A)\in A-T\cup\mathcal{D}(TP) \qquad \text{or} \qquad SR(A)\in T-Z.$$

Because the latter never continues infinitely by condition 2, the former, i.e. $SR(A)\in\Pi-T\cup\mathcal{D}(TP)$ must happen any desired time within finite iterations of step 2. This completes the proof.

Remember that X is allowed to be an infinite set. If we are not lucky enough to find a branching rule which generates only finite-length information se-

quences, Γ includes infinite-length ones. The infinite-length sequences correspond to infinite-length paths in a search tree. By the definition, Z can not include subproblems definied by them. Such subproblems must be included in $(T-Z)\cup\mathcal{D}(TP)$ if TP is finite. As mentioned in remark 7 of the preceding chapter, subproblems in $T-Z$ can be selected and branched from. Thus, finiteness in the sense of parallel operations(condition 3) is not enough. Tree programming is a sequential algorithm and additional conditions 1 and 2 must be satisfied. In other words, conditions 1 and 2 are needed to prevent tree programming from cyclig or zig-zag operations around $T\cup\mathcal{D}(TP)$, which may happen by chance. An example of zig-zag operations is shown in Fig. 3. In the figure, P_i is the i-th branched-from subproblem. Suppose that SR of a TP continues to select subproblems in the two tails starting from P_6 and P_7, and that these tails are of infinite length. Then, TP can not terminate, because the left most subproblem in $BR^2(P_0)$ has not been branched from yet and is not included in $T\cup\mathcal{D}(TP)$.

If T is larger than Z, TP is usually an approximate solution algorithm. Corollary 2 states that finiteness in the sense of parallel operations is sufficient for TP with $T=Z$ to be finite, and corollaly 3 tells us that testing satisfication of condition 3 under $T=Z$ is a cheap method for establishing finiteness of TP with T larger than Z.

Corollary 2. Let $T=Z$. If condition 3 holds true for any $\omega\in\Omega$, then TP is finite.

Proof: Immediate from theorem 2 and the fact that conditions 1 and 2 are trivially satisfied if $T=Z$.

Corollary 3. If the condition 3 holds true for $T=Z$ and for any $\omega\in\Omega$, then TP is finite for any T including Z.

Proof: Immediate from corollary 1 and 2.

Generally speaking, subproblems not in $\mathcal{D}(TP)$ can be eliminated and the condition 3 is not a necessary condition. But, if $\mathcal{D}(TP)$ is the set of subproblems possibly eliminated(see [22] for such cases), the condition is necessary for finiteness.

Theorem 3. Let TP satisfy condition 1, and assume P is eliminated if and only if $P\in\mathcal{D}(TP)$. Then, if $N(TP)$ is finite, condition 3 holds true.

Proof: Let $N(TP)=m$. Then, $A_{m+1}\subset T$. Define n_1 as

$$n_1=\max\{q \mid P\in BR^q(P_0),\ P\in A_{m+1}\}.$$

Let $\bar{A}=L_{m+1}-A_{m+1}$. By the assumption on $\mathcal{D}(TP)$, $\bar{A}\subset\mathcal{D}(TP)$. Define n_2 as

$$n_2=\max\{q \mid P\in BR^q(P_0),\ P\in\bar{A}\}.$$

By comparing the definitions of $N(TP)$ and $BR^q(P_0)$, it is understood that $n_1\leq$

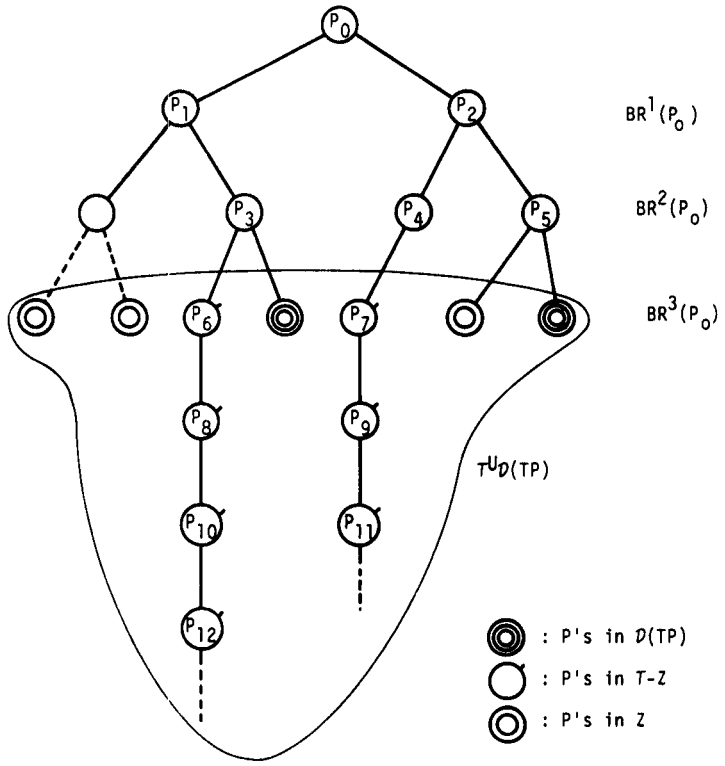

Fig. 3. A schematic representation of theorem 2.

$m+1$ and $n_2 \leq m+1$ must hold. Let $k=\max\{n_1, n_2\}$. By condition 1, we get

$$BR^k(P_0) \subset T \cup \mathcal{D}(TP).$$

By theorem 3, it is realized that the conditions in theorem 2 might be the weakest one. Notice also that condition 3 with $T=Z$ and $\mathcal{D}(TP)=\phi$ implies finiteness of $\gamma \in \Gamma$. If $|X|<+\infty$, condition 3 can be altered by condition 4.

[cond. 4] $(\forall P \in \Pi)(\exists n<+\infty)(\forall P' \in BR^n(P))\ ((X(P') \subsetneq X(P)) \vee (P' \in T \cup \mathcal{D}(TP)))$

If the part $X(P') \subsetneq X(P)$ were not in condition 4, it is essentially equivalent to condition 3. The next lemma implies that condition 4 together with condition 1 guarantees satisfication of condition 3 for subproblems with a finite domain of search.

Lemma 1. If conditions 1 and 4 hold true, for any $P \in \Pi$ it holds that

$$|X(P)|<+\infty \rightarrow (\exists m<+\infty)(BR^m(P) \subset T \cup \mathcal{D}(TP)).$$

Proof: Let P be an arbitrary one in Π. If $P \in Z$, the lemma trivially holds true for $m=0$. Assume $P \in \Pi - Z$. Condition 4 requires the existence of $n<+\infty$ such that

$$BR^n(P) \subset U(P) \cup T \cup \mathcal{D}(TP), \quad \text{where } U(P)=\{P' \mid |X(P')|=1 \text{ and } X(P') \subset X(P)\},$$

because $X(P)$ is a finite set. Again, by condition 4 every $P' \varepsilon U(P)$ must fall into $T \cup \mathcal{D}(TP)$ in finite iterations, i.e. there must be a finite v such that

$$BR^{v}(P') \subset T \cup \mathcal{D}(TP).$$

Let $m^{*}=\max\{v \mid P' \varepsilon U(P)\}$. m^{*} is well defined because $|BR^{n}(P)|<+\infty$ by B4. By condition 1, $BR^{q}(P) \subset T \cup \mathcal{D}(TP)$ for $q=n+m^{*}$. This completes the proof.

Notice that lemma 1 implies satisfication of condition 3 , if $X=X(P_{o})$ is a finite set. Therefore, conditions 1, 2 and 4 ensure finiteness. This fact is given in theorem 4 in a slightly generalized form.

Theorem 4. Assume conditions 1, 2 and 4 hold true for any $\omega \varepsilon \Omega$. If the next is true for any $\omega \varepsilon \Omega$, then TP is finite.

$$(\exists m<+\infty)\ (|\cup\{X(P) \mid P \varepsilon A_{m}\}|<+\infty). \tag{4.4}$$

Proof: (4.4) implies that $|X(P)|<+\infty$ is true for any $P \varepsilon A_{m}$. By lemma 1, we can define a finite $q(P)$ for each $P \varepsilon A_{m}$, where

$$q(P)=\min\{q \mid BR^{q}(P) \subset T \cup \mathcal{D}(TP)\}.$$

Notice also that $|A_{m}|<+\infty$ by B4. By doing almost the same as in the proof of theorem 2, $N(TP)<+\infty$ is established for each $\omega \varepsilon \Omega$.

Corollary 4. Let $|X|<+\infty$. If condition 4 holds true for $T=Z$ and any $\omega \varepsilon \Omega$, then, TP is finite for any T including Z.

Proof: If $|X|<+\infty$, TP' with $T=Z$ terminates in finite iterations under condition 4. (We can get a corollary of theorem 4 similar to corollary 2 of theorem 2.) Use corollary 1 in establishing finiteness for an arbitrary $(Z \subset) T$.

Now that the conditions for finiteness were clarified, we proceed to the correctness. Recall that equations (4.5) hold true for $i=0,1,\ldots,n(n=N(TP))$ by the definitions of ER and step 3 ii).

$$z \leqq \min\{u(P) \mid P \varepsilon A_{i}\} \qquad \text{for both } TP_{a} \text{ and } TP_{s}. \tag{4.5a}$$

$$\cup\{X(P) \mid P \varepsilon A_{i}\} \cap X^{*} \neq \phi \qquad \text{for } TP_{s}. \tag{4.5b}$$

$$X^{*} \subset \cup\{X(P) \mid P \varepsilon A_{i}\} \qquad \text{for } TP_{a}. \tag{4.5c}$$

Theorem 5. Let $TP=(SR,BR,u,ER,T=T_{1} \cup Z)$, then equations (4.6) hold true at the termination, where X_{1}^{*} is defined by (4.7).

$$X_{1}^{*} \neq X^{*} \rightarrow X^{*}-X_{1}^{*} \subset I \qquad \text{for } TP_{a}. \tag{4.6a}$$

$$X_{1}^{*} = X^{*} \rightarrow f(X(P_{0})) \leqq z \leqq \max\{u(P) \mid P \varepsilon T_{1}\} \qquad \text{for } TP_{a}. \tag{4.6b}$$

$$X_{1}^{*} = \phi \rightarrow I \cap X^{*} \neq \phi \qquad \text{for } TP_{s}. \tag{4.6c}$$

$$X_{1}^{*} \neq \phi \rightarrow f(X(P_{0})) \leqq z \leqq \max\{u(P) \mid P \varepsilon T_{1}\} \qquad \text{for } TP_{s}. \tag{4.6d}$$

$$X_{1}^{*}=\cup\{X(P) \mid P \varepsilon T_{1}\} \cap X^{*} \tag{4.7}$$

Proof: For TP_a, $A_n \subset T$ and (4.5c) imply that

$$X^*-X_1^* \subset \{X(P) \mid P\varepsilon A_n-T_1\} \text{ and } A_n-T_1 \subset Z.$$

By step 3 ii), all optimal solutions in A_n-T_1 must be conserved in I, i.e. (4.6a) must be true. $f(X(P_0)) \leqq z$ is trivial and if $X_1^*=X^*$ we may not get any optimal solutions before termination. But from (4.5a), it is true that

$$\begin{aligned} z &\leqq min[min\{u(P) \mid P\varepsilon A_n-T_1\},\ min\{u(P) \mid P\varepsilon A_n \cap T_1\}] \\ &\leqq \max\{u(P) \mid P\varepsilon T_1\}. \end{aligned}$$

This establishes (4.6b). (4.6c) and (4.6d) can be proved similarly.

It may seem peculiar that the condition for single optimal solution cases is more strict than the one for all optimal solution cases. As an explanation, remember that *ER* in TP_s may eliminate all but one optimal solutions. If the remaining optimal solution is in X_1^*, no optimal solution may be obtained. The next corollary is immediate from theorem 5 and gives a condition for correctness of tree programming.

Corollary 5. If $T=Z$, then tree programming is correct, i.e.

$$I=X^* \text{ for } TP_a \quad \text{and} \quad I \cap X^* \neq \phi \text{ for } TP_s.$$

Remarks on the results

1) Corollary 1 for $T'=Z$ gives a comparison of computational requirements among an exhaustive enumeration(TP), an implicit enumeration(TP') and a kind of approximate enumerations(TP"). Theorem 1 shows the tighter relations among them. See also remark 5, below.

2) The conditions 3 and 4 with $T=Z$ and $\mathcal{D}(TP)=\phi$ are the sufficient conditions for finiteness of exhaustive enumerations with an infinite and a finite domains of search, respectively. See corollaries 2 and 4.

3) As an example of theorem 2, *ER* in cutting plane methods(cf. example 6 in the next section) eliminates P if and only if $P\varepsilon\mathcal{D}(TP)$(=the set of infeasible problems). The proof of finiteness of the methods is completed by showing that P_0 falls into Z(=the set of linear programming problems having integer optimal solutions) in finite iterations of cutting plane generation.

4) The larger the size of $\mathcal{D}$-set is, the smaller the maximum computational requirement will be. Notice that eliminating subproblems in $\mathcal{D}$-set is equivalent to or more than enlarging the terminating condition from T to $T \cup \mathcal{D}(TP)$, and that theorem 1 indicates the tendency above. Because one of the difficulties of implicit enumeration algorithms is the tendency that they may reduce to exhaustive enumerations, the concept of $\mathcal{D}$-set is very important. $|\Pi-Z \cup \mathcal{D}(TP)|$ is the possible maximum value for $N(TP)$. If this cardinarity is small enough, tree programming can be a polynomial time algorithm. Johnson's rule for two

stage flow shop scheduling problem can be recognized as an example of this sort of tree programming.

5) Theorem 5 shows that introducing T_1 makes tree programming a sort of approximate algorithms. In cases of (4.6b) and (4.6d), if max $u(P)$ over T_1 is estimated exlicitly, tree programming is, so called, a guaranteed accuracy algorithm(see [14] and example 3 in the next section). If T_1 is made sure previously that it is the case of (4.6a) or (4.6c), it is natural to expect that the subproblems in T_1 can be included in $\mathcal{D}$-set and that T can be put $\mathcal{Z}$.

6) In all the tree programmings discussed up to now, T was equal to or larger than $\mathcal{Z}$. We can suppositionally regard I as a set of trivial problems. Notice that I always contains the best solutions in $A_{i\cap}\mathcal{Z}$. Hence, practically every subproblem in $\mathcal{Z}$ can be eliminated by using the lower bound tests(cf. example 3), and we can put $T=T_1$. But, to do this is to violate E2, and then E2 must be slightly modified. Notice that if this was done, $T=\phi$ in example 6, and tree programming can not include cutting plane methods, because their *ER*'s does not include the lower bound tests.

5. Illustrative examples

Branch-and-bound mehtods, the additive implicit enumeration, cutting plane mehtods and dynamic programming algorithm are reformulated as special kinds of tree programming. The purpose here is to show the universality of tree programming concept over a wide variety of algorithms. See example 2 for an exhaustive enumeration, where *ER* is an identity mapping.

Models in [5] and [17] do not contain other dominance tests than lower bound tests. They can not represent example 4. The model in [13] do not include feasibility tests and can not explain example 6. Moreover, this model is only for permutation problems. Ibaraki's model seems most flexible. But, it can explain neither analytical solutions(cf. remark 4 in the preceding section) nor a guaranteed accuracy algorithm in example 3. The method in [12] will be explained within tree programming framework, too. Example 5 can be explained by any one of these models. This algorithm has many variants, which are usually distinguished from branch-and-bound methods(e.g. see p. 210 of [20]).

Example 3. Branch-and-bound methods

In recent papers[10,13,14], a generalized or abstracted version of the hybrid dynamic programming/branch-and-bound algorithm(DP/BB) in example 4 is also called "branch-and-bound method". Such version is also regarded as tree programming(cf.[24]). Here, we use the term "branch-and-bound" as the one in

the sense of Little et al.[16], Balas[3], Mitten[17] and Ibaraki[9].

A lower bounding function is a mapping $g : \Pi \rightarrow E$ such that

$$g(P) \begin{cases} = f(X(P)) & \text{for } P \in Z \\ \leq f(X(P)) & \text{otherwise.} \end{cases} \tag{5.1}$$

An upper bounding function of the second type is $u_g : \Pi \rightarrow E$, such that

$$u_g(P) \begin{cases} = f(X(P)) & \text{for } P \in Z \\ \geq f(X(P)) & \text{otherwise.} \end{cases} \tag{5.2}$$

Let $z_g = +\infty$ at step 1, and assume that z_g is updated by (5.3) after each branching operation, somewhere between step 3 i) and step 4.

$$z_g = \min[z_g, \min\{u_g(P) \mid P \in A_i\}]. \tag{5.3}$$

Obviously, $f(X(P_0)) \leq z_g$ and $g(P) \leq f(X(P_0))$ hold true. Then, an elimination rule which consists of a lower bound test can be defined as a mapping $ER : E \times \Lambda \rightarrow \Lambda$ such that if $ER(t,A)=A'$, then

$$((\forall P \in A')\ (g(P) \geq z_g)) \wedge ((\forall P \in A - A')\ (g(P) > z_g)) \qquad \text{for } ER_a$$

$$((\forall P \in A' - P^*)\ (g(P) < z_g)) \wedge ((\forall P \in A - A')\ ((g(P) \geq z_g) \wedge (P \notin P^*))) \qquad \text{for } ER_s,$$

where P^* is a singleton set of a subproblem which has given the current value z_g.

As simple upper bounding functions, we can use the following one.

$$u(P) = u_g(P) = \begin{cases} g(P) & \text{for } P \in Z \\ +\infty & \text{otherwise.} \end{cases} \tag{5.4}$$

Then, it is easily realized that TP with ER defined above (notice that $z = z_g$ in this case) is a branch-and-bound method in the sense used here. The definition (5.1) is similar to the one in [9] and [17]. Lower bounding functions are often restricted by the third condition;

$$g(P) \leq g(P') \qquad \text{if } X(P') \subset X(P).$$

Making g satisfy this condition is very easy in practice and it makes ER more powerful. But this condition is not needed for the object here.

In order to get a guaranteed accuracy algorithm, let, as an instance[14],

$$T = Z \cup \{P \in \Pi \mid g(P) \geq z(1-\xi),\ 0 \leq \xi < 1\}. \tag{5.5}$$

It is known also that the better the initial incumbent is, the less the computational requirement is[14].

Example 4. A hybrid DP/BB algorithm[18]

We consider a discrete multistage decision problem with an additive objective function. S is the finite set of all feasible states. D is the finite set of all possible decisions. T: $S \times D \rightarrow S$ is a transition mapping, which means if a decision $d \in D$ is implemented on a stage with a state $s \in S$, the state on the next stage is $T(s,d)$. $A(s)=\{d \in D \mid T(s,d) \in S\}$ is the set of all feasible decisions at a state s. γ is a sequence of decisions; $\gamma=(d_1,d_2,\ldots,d_n)$, $1 \leq n$. For simplicity, if a state s' results from γ implemented in its sequence on a

state s, denote $T(s,\gamma)=s'$. $c(s,d)$ represents an incremental cost when d is implemented on a state s. $s_0 \in S$ is an initial state and $S_e \subset S$ is the set of feasible final states. Then, a data which describes a problem is the set $\omega=(s_0,S,S_e,D,T,c)$. Let X and F be the set of feasible decisions which make moves from s_0 to $s \in S-\{s_0\}$ and from s_0 to $s \in S_e$, respectively, i.e.

$$X=\{\gamma \mid T(s_0,\gamma)\in S-\{s_0\}, \text{ and } T(s_{i-1},d_i)=s_i \in S \text{ for } i=1,2,\ldots,n\},$$

$$F=\{\gamma \mid T(s_0,\gamma)\in S_e, \text{ and } T(s_{i-1},d_i)=s_i \in S \text{ for } i=1,2,\ldots,n\}.$$

The cost of $x \in X$ can be represented by f; assume $x=(d_1,d_2,\ldots,d_n)$, then

$$f(x)=\sum_{i=1}^{n} c(s_{i-1},d_i), \qquad \text{where } T(s_{i-1},d_i)=s_i \text{ for } i=1,2,\ldots,n .$$

The problem actually solved by the algorithm below is (2.2) with the parameters defined above.

$\gamma\cdot\gamma'$ is a decision sequence where γ is followed by γ'. Let $\bar{F}(\gamma)$ be the set of decision sequences which can be concatinated to γ; $\bar{F}(\gamma)=\{\gamma' \mid \gamma\cdot\gamma' \in X\}$. Then, $X(P(\omega,\gamma))$ is the set $\{\gamma\cdot\gamma' \mid \gamma' \in \bar{F}(\gamma)\}$. A lower bounding function is defined as

(5.6) $$g(P(\omega,\gamma))=f(\gamma)+\bar{f}(\gamma), \quad \text{where}$$

(5.7) $$\bar{f}(\gamma)\begin{cases} \leq \min\{f(\gamma') \text{ under a initial state } s_0' \mid \gamma' \in \bar{F}(\gamma),\ s_0'=T(s_0,\gamma)\} & \text{if } \gamma \notin F, \\ = 0 & \text{otherwise.}\end{cases}$$

We can put $Z=\{P(\omega,\gamma) \mid \gamma \in F\}$. Define $u(P)$ and $u_g(P)$ by (5.4), then g, u and u_g satisfy (5.1), (3.3) and (5.2), respectively. Define the constituents of TP other than u as follows.

SR : FIFO rule, i.e. select the first generated subproblem in A_i-Z. Let $P_i(\omega,\gamma)$ be the selected one.

BR : $BR(P_i(\omega,\gamma))=\{P(\omega,\gamma\cdot d) \mid d \in A(s), \text{ where } s=T(s_0,\gamma)\}$.

ER : If $ER(t,A)=A'$, then

(5.8) $$A-A'=\begin{cases} \{P(\omega,\gamma)\in A \mid (g(P(\omega,\gamma))>z) \\ \quad \forall((\exists P(\omega,\gamma')\in A)((T(s_0,\gamma)=T(s_0,\gamma'))\wedge(f(\gamma)>f(\gamma')))\} & \text{for } TP_a \\ \{P(\omega,\gamma)\in A-P^* \mid (g(P(\omega,\gamma))\geq z) \\ \quad \forall((\exists P(\omega,\gamma')\in A')((T(s_0,\gamma)=T(s_0,\gamma'))\wedge(f(\gamma)\geq f(\gamma')))\} & \text{for } TP_s. \end{cases}$$

$T=Z$.

SR and *BR* above trivially satisfy S1, S2 and B1 through B4, respectively. *ER* satisfies E1. The remaining to be shown is the satisfication of E2. Notice that *SR* selects a subproblem with the shortest decision sequence and that γ' of P in $BR(P(\omega,\gamma))$ is longer than γ just by one. This means that there appear repeatedly A_i's in which all related γ's have the same length k. Let $f_k(s)$ be the minimum cost for bringing the system from s_0 to $s \in S$ by at most k decisions.

At the instance mentioned above, the following equation must hold for each $P(\omega,\gamma)\in A_i$.

$$(5.9)\qquad f_k(s)=f(\gamma)=\min\{f(\gamma') \mid s=T(s_0,\gamma)=T(s_0,\gamma'),\ |\gamma'|=|\gamma|=k\}$$
$$=\min\{f_{k-1}(s')+c(s',d) \mid d\in A(s'),\ s=T(s',d),\ s'\in S\}.$$

(5.9) is the functional equation of dynamic programming. Thus, *ER* satisfies E2 and TP with the parameters defined here is really tree programming, and this algorithm is called a hybrid DP/BB [18]. Notice that this algorithm reduces to a branch-and-bound method if the last half of (5.8) is omitted, and to a dynamic programming algorithm if $z\longrightarrow+\infty$(i.e. the lower bound test is not active). The model in [10,13,14] are regarded as general versions of this algorithm.

Example 5. The additive implicit enumeration[4]

The additive implicit enumeration(AIE) algorithm is one for solving 0-1 integer linear programming problems. Define notations as follows.

$N=\{1,2,\ldots,n\}$, $M=\{1,2,\ldots,m\}$, $x=(x_1,x_2,\ldots,x_n)^t$, $c=(c_1,c_2,\ldots,c_n)$ where $c_q(q\in N)$ are nonnegative integers, $b=(b_1,b_2,\ldots,b_m)^t$ where $b_p(p\in M)$ are integers and $A=\|a_{pq}\|$ where $a_{pq}(p\in M,\ q\in N)$ are integers. Let $X=\{x \mid x_q=0 \text{ or } 1 \text{ for } q\in N\}$, $f(x)=cx$ and $F=\{x\in X \mid Ax\geq b\}$. Then the problem solved by AIE is defined by (2.2) with the parameters defined above. ω here is (N,M,A,b,c). Let $P_i(\omega,\gamma_i)$ be the subproblem selected as the i-th branched-from one. Here, γ_i is the branching history and is defined as the union $\gamma_i^1\cup\gamma_i^0$, where

γ_i^1 : the set of $q\in N$ already fixed at $x_q=1$,

γ_i^0 : the set of $q\in N$ already fixed at $x_q=0$.

The sequence of the elements in γ_i is not important(but is determined by *SR*), and we will treat γ_i as a set for convenience sake. $N_i=N-\gamma_i$ is the set of free variables. The domain of search of $P_i(\omega,\gamma_i)$ is

$$X(P_i(\omega,\gamma_i))=\{x\in X \mid x_q=1 \text{ for } q\in\gamma_i^1,\ x_q=0 \text{ for } q\in\gamma_i^0\}.$$

We can put $T=Z=\{P(\omega,\gamma_i) \mid$ the 0-completion of γ_i is feasible$\}$, where the 0-completion of γ_i is x such that $x_q=1$ for $q\in\gamma_i^1$, $x_q=0$ for $q\in N-\gamma_i^1$. A lower bounding function is defined as

$$(5.10)\qquad g(P_i(\omega,\gamma_i))=\sum_{q\in\gamma_i^1}c_qx_q\begin{cases}=f(X(P_i(\omega,\gamma_i))) & \text{for } P\in Z,\\ \leq f(X(P_i(\omega,\gamma_i))) & \text{otherwise.}\end{cases}$$

Define u and u_g by (5.4), then g, u and u_g satisfy (5.1), (3.3) and (5.2), respectively.

SR : LIFO rule, i.e. select the last generated subproblem in A_i-Z.

In defining the remaining constituents, notations below are needed.

$$d_p=\sum_{q\in N_i-Q_c^-}\max\{0,a_{pq}\}+\sum_{q\in\gamma_i^1}a_{pq}-b_p \qquad \text{for } p\in M,$$

$$\text{where} \quad Q_c^-=\{j\in N_i \mid c_j\geq z-\sum_{q\in\gamma_i^1}c_q\}.$$

If $d_p\geq 0$ for all $p\in M$, make Q_i^+ and Q_i^- as follows.

$$Q_i^+=\{j\in N_i-Q_c^- \mid a_{pj}>d_p,\ p\in M\}, \qquad Q_i^-=\{j\in N_i-Q_c^- \mid -a_{pj}>d_p,\ p\in M\}\cup Q_c^-.$$

Moreover, if $Q_i^+=Q_i^-=\phi$, select j^* which satisfies

$$V_{j^*}=\min\{V_j \mid j\in N_i\}, \quad \text{where} \quad V_j=\sum_{p\in M}\max\{0,\ b_p-\sum_{h\in\gamma_i^1}a_{ph}-a_{pj}\}.$$

Notice that there is no feasible completion of γ_i if $d_p<0$ for some $p\in M$, that $x_q=1(q\in Q_i^+)$, $x_q=0(q\in Q_i^-)$ are necessary for completions of γ_i to be feasible and that j^* can be regarded as an index for which $x_{j^*}=1$ added to γ_i makes γ_i near to the feasible one most effectively among the free variables. Thus, we can define *BR* and *ER* as follows.

BR : $BR(P_i(\omega,\gamma_i))=\{P_\alpha,P_\beta\}$, where

$$\begin{cases} P_\alpha : \gamma_\alpha^1=\gamma_i^1\cup Q_i^+, \quad \gamma_\alpha^0=\gamma_i^0\cup Q_i^- \\ P_\beta : \text{one such that } X(P_\beta)=X(P_i)-X(P_\alpha) \end{cases} \qquad \text{if } Q_i^+\cup Q_i^-\neq\phi$$

$$\begin{cases} P_\alpha : \gamma_\alpha^1=\gamma_i^1\cup\{j^*\}, \quad \gamma_\alpha^0=\gamma_i^0 \\ P_\beta : \gamma_\beta^1=\gamma_i^1, \quad \gamma_\beta^0=\gamma_i^0\cup\{j^*\} \end{cases} \qquad \text{if } Q_i^+\cup Q_i^-=\phi.$$

ER : If $ER(t,A)=A'$ where $A=(A_i-\{P_i\})\cup BR(P_i)$, then

$$A-A'=\begin{cases} \{P\in BR(P_i) \mid ((\exists p\in M)(d_p<0))\vee(Q^+\cap Q^-\neq\phi)\} \\ \qquad \cup\{P_\beta \mid Q_i^+\cup Q_i^-\neq\phi\}\cup\{P\in A-P^* \mid g(P)\geq z\} & \text{for } TP_s, \\ \{P\in BR(P_i) \mid ((\exists p\in M)(d_p<0))\vee(Q^+\cap Q^-\neq\phi)\} \\ \qquad \cup\{P_\beta \mid Q_i^+\cup Q_i^-\neq\phi\}\cup\{P\in A \mid g(P)>z\} & \text{for } TP_a. \end{cases} \tag{5.11}$$

BR and *ER* also satisfy B1 through B4 and E1, E2, respectively. Algorithm TP with the parameters defined above is tree programming. AIE in [4] is different from this algorithm in the point that the test for elimination is done just after step 2 only on P just selected. Notice that the middle part in the right hand side of (5.11) is a formality, because P_β in the case may not be generated practically. Notice also that all information necessary for *ER*, except the middle part, can be made just from γ of each problem. Therefore, a subproblem can be eliminated in AIE if and only if it is eliminated in TP. Thus, TP is essentially AIE in the sense that the sequence of actually branched subproblems in TP is exactly the same as one in AIE, i.e. the STS's corresponding to them are essentially the samething(cf. remark 1 in section 3 and [24]).

Example 6. Cutting plane methods

Algorithms for pure integer linear programming problems are treated below. Define N, M, a, b, c, x and f as the same as in example 5. F and X are defined as

$$F=\{x \mid Ax\geqq b,\ x_q\text{: nonnegative integer for } q\in N\},$$
$$X=\{x \mid Ax\geqq b,\ x_q\text{: nonnegative real for } q\in N,\ cx\leqq cx^*\}.$$

Again, the problem actually solved by the algorithm below is defined by (2.2) with the parameters defined above. Let P_i be a subproblem of $P_0(\omega)$. R_i denotes the set of feasible solutions of the linear programming problem corresponding to P_i. For example,

$$R_0=\{x \mid Ax\geqq b,\ x_q\text{: nonnegative real for } q\in N\}.$$

Suppose that a cutting plane can be produced by a proper method. Cutting planes are denoted by $vx=w$, where v is a $1\times n$ vector and w is a scalor. P_i is divided into P_α and P_β where $X(P_\alpha)=\{x\in X(P_i) \mid vx\geqq w\}$ and $X(P_\beta)=\{x\in X(P_i) \mid vx<w\}$. Then, $X(P_\beta)\cap F=\phi$ from the property of cutting planes and $R_\alpha=\{x\in R_i \mid vx\geqq w\}$. A cutting plane method is TP with the following constituents.

SR : An identical mapping(A_i is always a sigleton set).

BR : $BR(P_i)=\{P_\alpha,P_\beta\}$.

u : $u(P)=\begin{cases}\min\{cx \mid x\in X(P)\} & \text{if } P\in Z\\ +\infty & \text{otherwise.}\end{cases}$

ER : $ER(t,\{P_\alpha,P_\beta\})=\{P_\alpha\}$.

$T=Z=\{P \mid R \text{ of } P \text{ has optimal integer vertices}\}$.

Notice that cutting plane methods search for optimal solutions from the outside of the feasible region, and that R used for generating a cutting plane is not the domain of search for a cutting plane method. Research on how to make deeper cutting planes should be recognized as one for better BR and ER.

6. Conclusion

Tree programming has been proposed as a general framework unifying the conventional models. The essential difference from the conventional ones is in the point that the elimination rule is defined as any mechanism which eliminates some of active subproblems. The elimination rule will be constructed by combining four test functions[22]. Because of this flexibility, various algorithms were shown to be sorts of tree programming. The conditions for finiteness and correctness were obtained for both infinite and finite domains of search. They were given in simple forms and clarify the properties which each constituent must have.

The results in this paper are expected to be used as guide lines in designing enumerative algorithms. But, they are elemental conditions and how to compose various candidates of each constituent so as to get efficient algorithms should be explored. This subject is discussed in [22]. Elimination rules must be given as a procedure, so as to do analysis similar to those in [10,12,13,14]. An elimination procedure satisfying E1 and E2 is shown in [24], which is specific enough to do useful analysis and general enough to explain more specific models proposed in these papers.

Ackowledgment

The author wishes to thank the members of a research group, CORG, especially Dr. A. Ōuchi and professor H. Kubo of Hokkaido University, and professor N. Wakabayashi of Otaru University of Commerce, for their helpful discussions. He also wishes to thank professor T. Ibaraki of Kyoto University, who pointed out the possibility of including the cases where problems may not be decomposed into disjoint subproblems.

This reseach was supported in part by grant in aid for scientific research of the Ministery of Education, Science and Culture in Japan under grant: syorei kenkyu (A) 173018.

References

[1] Agin, N.: Optimum Seeking with Branch and Bound. *Management Science*, Vol. 13, No 4(1966), B176-B185.

[2] Baker, K.R.: *Introduction to Sequencing and Scheduling*. John Wiley and Sons, 1974.

[3] Balas, E.: A Note on the Branch and Bound Principle. *Operations Research*, Vol. 16, No. 2(1968), 442-445.

[4] Balas, E.: Implicit Enumeration; An Introductory Survey. Part I in *Integer and Combinatorial Programming*. Jyohosyori Kensyu Center, 1975.

[5] Geoffrion, A.M. and Marsten, R.E.: Integer Programming Algorithms; A Framework and State of the Art Survey. *Management Science*, Vol. 18, No. 9(1972), 465-491.

[6] Golomb, S.M. and Baumert, L.D.: Backtrack Programming. *Journal of the Association for Computing Machinery*, Vol. 12, No. 4(1965), 516-524.

[7] Held, M. and Karp, R.M.: A Dynamic Programming Approach to Sequencing Problems. *SIAM Journal*, Vol. 10, No. 1(1962), 196-210.

[8] Hu, T.C.: *Integer Programming and Network Flows*. Addison-Wesley, 1969.

[9] Ibaraki, T.: Theoretical Comparison of Search Strategies in Branch-and-Bound Algorithms. *International Journal of Computer and Information Sciences*, Vol. 5(1976), 315-344.

[10] Ibaraki, T.: The Power of Dominance Relations in Branch-and-Bound Algorithms. *Journal of the Association for Computing Machinery*, Vol. 24, No. 2(1977), 264-279.

[11] Ibaraki, T.: Why is Integer Programming Difficult(in Japanese). *Mathematical Programming, Fifth Symposium held by Operations Research Society of Japan*, March, 1977.

[12] Ibaraki, T.: Computational Efficiency of Approximate Branch-and-Bound Algorithms. *Mathematics of Operations Research*, Vol. 1, No. 3(1976), 287-298.

[13] Kohler, W.H. and Steiglitz, K.: Characterization and Theoretical Comparison of Branch-and-Bound Algorithm for Permutation Problems. *Journal of the Association for Computing Machinery*, Vol. 21, No. 1(1974), 140-156.

[14] Kohler, W.H. and Steiglitz, K.: Enumerative and Iterative Computational Approaches. Chapter 6 in *Computer and Job-Shop Scheduling Theory*, Coffman,Jr., E.G.(ed.), John Wiley and Sons, 1976.

[15] Lawler, E.L. and Wood, D.E.: Branch-and-Bound Methods: A Survey. *Operations Research*, Vol. 14, No. 4(1966), 669-719.

[16] Little, J.D.C., Murty, K.G., Sweeny, D.W. and Karel, C.: An Algorithm for the Travelling Salesman Problem. *Operations Research*, Vol. 11, No. 12(1963), 972-989.

[17] Mitten, L.G.: Branch-and-Bound Methods: General Formulation and Properties. *Operations Research*, Vol. 18, No. 1(1970), 24-34.

[18] Morin, T.L. and Marsten, R.E.: Branch-and-Bound Strategies for Dynamic Programming. *Operations Research*, Vol. 24, No. 4(1976), 611-627.

[19] Müller-Merbach, H.: Modeling Techniques and Heuristics for Combinatorial Problems. pp. 3-27 in *Combinatorial Programming; Methods and Applications*, Roy, B.(ed.), D. Reidel Publishing, 1975.

[20] Salkin, Harvey M.: *Integer Programming*. Addison-Wesley, 1975.

[21] Sekiguchi, Y.: Branch-and-Bound Algorithm(in Japanese). Part II in *Kodo Sentaku no Tame no Suiron*, Nagao, T.(ed.), Kōseisha Kōseikaku, 1972.

[22] Sekiguchi, Y.: Relations between Effectiveness and Constituents of Tree Programming. Manuscript, 1977.

[23] Sekiguchi, Y.: Expanded Backtrack Programming. *Proceedings of CAM-I International Congress, Hamilton, Ontario, Canada*(1974), 18-38.

[24] Sekiguchi, Y.: Equivalency of Models of Enumerative Algorithms. Manuscript, 1978.

[25] Wells, M.B.: *Elements in Combinatorial Computing*. Chapter 4. Pergamon Press, 1971.

Yasuki SEKIGUCHI: Department of Business Administration, Faculty of Economics and Business Administration Hokkaido University, Kitaku Kita 8 Nishi 7, Sapporo, 060, Japan.