Stochastic Modeling of Branch-and-Bound Algorithms with Best-First Search

BENJAMIN W. WAH, MEMBER, IEEE, AND CHEE FEN YU, STUDENT MEMBER, IEEE

Abstract—Branch-and-bound algorithms are organized and intelligently structured searches of solutions in a combinatorially large problem space. In this paper, we propose an approximate stochastic model of branch-and-bound algorithms with a best-first search. We have estimated the average memory space required and have predicted the average number of subproblems expanded before the process terminates. Both measures are exponentials of sublinear exponent. In addition, we have also compared the number of subproblems expanded in a best-first search to that expanded in a depth-first search. Depth-first search has been found to have computational complexity comparable to best-first search when the lower-bound function is very accurate or very inaccurate; otherwise, best-fit search is usually better. The results obtained are useful in studying the efficient evaluation of branch-and-bound algorithms in a virtual memory environment. They also confirm that approximations are very effective in reducing the total number of iterations.

Index Terms—Approximations, best-first search, branch-and-bound algorithms, depth-first search, iterations, memory space, subproblem.

I. Introduction

THE search for solutions in a combinatorially large problem space is important in artificial intelligence and operations research. Search problems can be classified as either decision or optimization problems [7]. In a decision problem, one attempts to determine the existence of at least one solution that satisfies a given set of constraints. Examples include theorem-proving, expert systems, and some permutation problems. An optimization problem is characterized by an objective function to be minimized or maximized and a set of constraints to be satisfied. Examples include the traveling-salesman, warehouse-location, job-shop-scheduling, knapsack, vertex-cover, and integer-programming problems.

A combinatorial search can be put into the form of a constrained optimization:

$$\begin{aligned} &\text{Minimize } C_0(x) && x \in X \\ &\text{subject to } g_i(x) \leqslant 0 && i = 1, 2, \cdots, m \end{aligned} \tag{1}$$

in which X represents the domain of optimization defined by the m constraints, normally a Euclidean n-space, and x denotes a vector $(x_1, x_2, \cdots, x_n)$. A solution vector that lies in x is called a *feasible solution*, and a feasible solution for which $C_0(x)$ is minimal is called an *optimal solution*.

A general technique for solving combinatorial-search problems is the *branch-and-bound (B&B) algorithm* [18], [22]. A B&B algorithm is a partitioning algorithm that decomposes a problem into smaller subproblems and repeatedly decomposes the subproblems until infeasibility is proved or a solution is found. Many theoretical properties of serial B&B algorithms have been developed [9]-[12], [15], [24]. B&B algorithms have been applied to solve problems in scheduling [19], "knapsack" [13], "traveling salesman" [6], facility allocation [4], integer programming [8], and many others. It has been recognized [16] that B&B algorithms are a generalization of many heuristic search procedures such as A* [23], AO* [21], SSS* [26], B* [1], alpha-beta [14], and dynamic programming [2]. Parallel computers for evaluating B&B algorithms have also been studied [27], [35].

The state of the partitioning process in a B&B algorithm can be represented as a partial tree. A terminal node in the tree represents a partition and is called a *subproblem*. The partitioning process selects a partition and breaks it up into smaller partitions. As a result, the node representing this partition is extended by one level, and the children are used to denote the smaller partitions. As the branching process proceeds, subproblems that will not lead to an optimal solution are eliminated from further consideration. Eventually, the process terminates when better solutions cannot be found. B&B algorithms are, therefore, characterized by four constituents: branching rule(s), selection rule(s), elimination rule(s), and termination condition(s). The first two rules are used to decompose problems into simpler subproblems and appropriately order the search. The last two rules are used to to eliminate generated subproblems that are not better than the ones already known.

In a B&B algorithm, let P_i be a subproblem, and let $f(P_i)$ be the value of the best solution obtained by evaluating all the subproblems decomposable from P_i. Each subproblem is characterized by a value that is computed from a lower-bound function g. The lower-bound function satisfies the following properties:

a) g is a lower-bound estimate of f;

b) g is exact when P_i is feasible;

c) lower bounds of descendant nodes always increase.

The lower-bound function designed is highly dependent on the problem. For example, in an integer-programming problem, a linear program with relaxed integer constraints can be used [17]; in a traveling-salesman problem, an assignment algorithm [3] or a spanning-tree algorithm can be used.

The lower bound is calculated for a subproblem when it is

Manuscript received November 8, 1982; revised April 17, 1985. This work was supported in part by the National Science Foundation under Grant ECS81-05968 and by CIDMAC, a research unit of Purdue University, sponsored by Purdue, Cincinnati Milicron Corporation, Control Data Corporation, Cummins Engine Company, Ransburg Corporation, and TRW.

B. W. Wah is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

C. F. Yu is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0098-5589/85/0900-0922$01.00 © 1985 IEEE

created. If a subproblem is a feasible solution with the best objective-function value so far, the solution value becomes the *incumbent* z. In minimization problems, if the lower bound of a subproblem P_i exceeds the incumbent, P_i can be pruned because it will not lead to a better solution value than the incumbent. In other words, P_i is terminated during the computation if

$$g(P_i) \geqslant z. \tag{2}$$

The process continues until all subproblems are either expanded or eliminated.

The above elimination rule for obtaining an exact optimal solution can be relaxed in order to obtain a suboptimal solution with a guaranteed accuracy [18]. Suppose it were decided that a deviation of 10 percent from the optimum was tolerable. If a feasible solution of 150 is obtained, all subproblems with lower bounds of 136.4 (or 150/(1 + 0.1)) or more can be terminated, since they cannot lead to a solution that deviates by more than 10 percent from 150. This technique significantly reduces the amount of intermediate storage and time needed to arrive at a suboptimal solution. Define an *allowance function* $\epsilon(z)$: $R \to R$ (set of reals) such that P_i is terminated if

$$g(P_i) \geqslant z - \epsilon(z). \tag{3}$$

The final incumbent z_F obtained by using the modified lower-bound test deviates from the optimal solution value z_0 by [9]

$$z_F - \epsilon(z_F) \leqslant z_0 \leqslant z_F. \tag{4}$$

Examples of often-used allowance functions are

$$\epsilon(z) = \epsilon \qquad \epsilon \geqslant 0 \text{ (absolute error deviation)} \tag{5}$$

$$\epsilon(z) = \frac{\epsilon z}{1+\epsilon} \qquad \epsilon \geqslant 0, z \geqslant 0 \text{ (relative error deviation).} \tag{6}$$

The *selection rule* examines the list of active subproblems and selects one for expansion. If the list is maintained in a first-in/first-out order, the algorithm is called a *breadth-first search*. If the list is maintained in a last-in/first-out order, the algorithm is called a *depth-first search*. Lastly, if the list is maintained in increasing order of lower bounds, the algorithm is called a *best-first search*.

Once a subproblem has been selected for partitioning, the *branching rule* creates multiple subproblems by heuristically selecting some unassigned parameters in the subproblem and assigning alternatives for these parameters. For example, in the traveling-salesman problem, the unassigned parameters are the set of untraversed edges. In expanding a subproblem, an untraversed edge (i, j) is selected, and two alternatives are created: a) the edge is traversed and the salesman goes directly from City i to City j, and b) vice versa.

The B&B algorithm for minimization problems is summarized as follows:

```
incumbent z = ∞;
list of subproblems = {P0};
while list of subproblems ≠ ∅ do [
    apply selection rule to list of subproblems;
    expand selected subproblem by the branching rule;
    for P ∈ {children generated} do [
        if P is a feasible solution then [
            update incumbent z;
            apply elimination rule ]
        else if (g(P) < (z - ε(z))) then
            insert P into list of subproblems
    ]
]
```

To illustrate the B&B algorithm, the evaluation of an integer-programming problem [22] is shown here. Integer-programming problems can be expressed as a constrained optimization:

$$\begin{aligned} &\text{Minimize } CX \\ &\text{subject to } AX \geqslant B \\ &\qquad X^T = (x_1, x_2, \cdots, x_n); \\ &\qquad x_i\text{: nonnegative integer, } i = 1, 2, \cdots, n; \\ &\qquad A, B, \text{ and } C \text{ are constant matrices.} \end{aligned} \tag{7}$$

These problems differ from ordinary linear-programming problems in that the variables are restricted to nonnegative integer values.

One approach to the problem is the following. Apply the dual simplex method to a subproblem and solve it as a linear program. If the optimal solution is integral, a feasible solution has been generated; otherwise, create two new subproblems as follows. Choose a variable in the subproblem that has a nonintegral value (say $x_i = 4.4$) and restrict that variable to the next lower integral value for one subproblem ($x_i \leqslant \lfloor 4.4 \rfloor$ or $x_i \leqslant 4$) and to the next higher integral value ($x_i \geqslant \lceil 4.4 \rceil$ or $x_i \geqslant 5$) for the other. The variable chosen is the one with the greatest up or down penalty. The up penalty for a variable x_i, having a value of a_i, is the estimate of the amount by which the solution to the current subproblem would increase if the integral constraint $x_i \geqslant \lfloor a_i \rfloor$ was introduced. The down penalty is similar, except that it is associated with the constraint $x_i \leqslant \lceil a_i \rceil$. The lower bound of a new subproblem is the sum of the optimal simplex solution and the associated penalty. This entire process is repeated on the new subproblems.

Fig. 1(b) shows the B&B tree for the problem in Fig. 1(a). The dual simplex method gives an optimal solution of 14.2 for the original problem. Since the variables are not integral, a feasible solution has not been generated. Up and down penalties are calculated for the variables. x_1 has the greatest penalty ($U = 1.8$). Two new subproblems are then created, one with $x_1 = 0$, and the other with $x_1 \geqslant 1$. The evaluated lower bounds are shown in Fig. 1(b). The dual simplex method is applied again to the subproblem with the smallest lower bound, and a feasible solution is generated in which all the variables are integral. This constitutes an optimal solution since the lower bound of the remaining subproblem is greater.

It has been shown that a best-first search has a predictable and the best time efficiency if $g(P_i) \neq f^*$, in which P_i is any node other than an optimal-solution node and f^* is the optimal-solution value [20]. From simulations, it was found that the best-first search requires the minimum execution time but a large amount of memory space. On the other hand, in a depth-first search, the list of subproblems is stored in a last-in/first-out stack with size equal to the height of the search tree. For many search problems, this height is equal to the number of

$$\min x_0 = 7x_1 + 3x_2 + 4x_3$$
$$x_1 + 2x_2 + 3x_3 \geq 8$$
$$3x_1 + 2x_2 + x_3 \geq 5$$
$$x_1, x_2, x_3 \geq 0, \text{ integer}$$

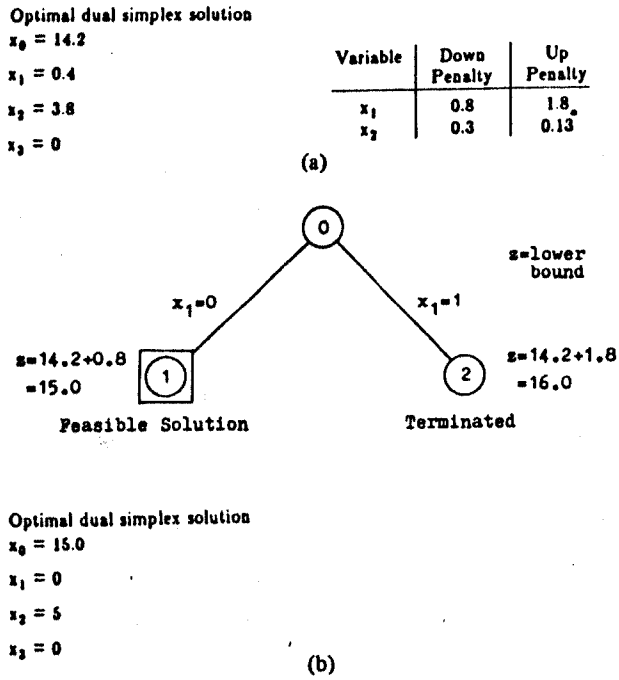


Fig. 1. (a) An example of an integer-programming problem. (b) The corresponding B&B solution.

variables in the problem and is reasonably small. A depth-first search, thus, has a small memory-space requirement. However, the time efficiency is unpredictable and can be much worse than that of a best-first search. Hence, the savings in space may be offset by the increased number of subproblems examined. A breadth-first search would have similar behavior.

In this paper, we present an approximate stochastic model of the B&B algorithm with a best-first search. We characterize the search process as two walls moving towards each other. From this model, we can derive the average termination time and the average memory-space requirements of a best-first search. We have also compared the number of subproblems expanded under a best-first search to that of a depth-first search. The results obtained have been applied to design a virtual-memory operating system for supporting B&B algorithms [28], [29]. Furthermore, the model is useful in studying the behavior of approximate B&B algorithms [20].

II. Model of B&B Process with a Best-First Search

A best-first search can be modeled as two walls moving towards each other (Fig. 2). The front wall on the left represents the minimum lower bound of the currently active subproblems, and the back wall on the right represents the incumbent. Initially, the position of the front wall is undefined, and the back wall is at infinity. The lower bound of the problem is evaluated and is taken to be the position of the front wall. The problem is decomposed into multiple subproblems, and a lower bound is calculated for each subproblem. The front wall then moves to the minimum position of the set of active

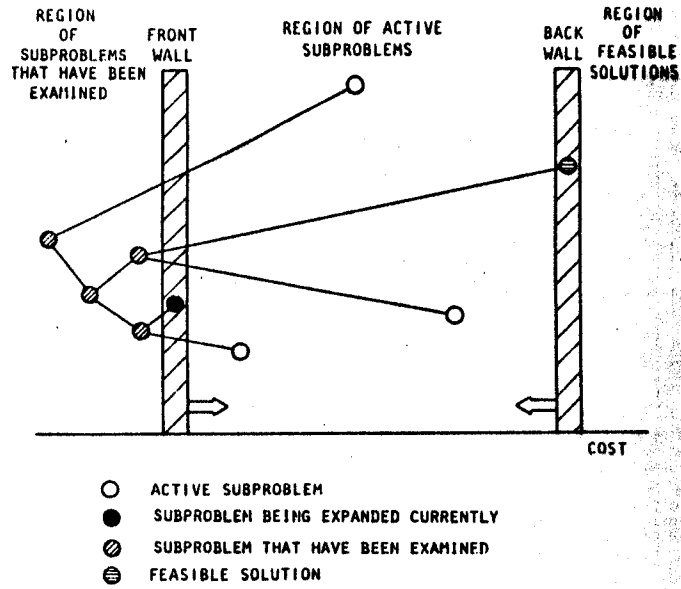


Fig. 2. Model of branch-and-bound process with a best-first search.

subproblems. Since the lower bounds of descendent subproblems are always greater than that of ancestor subproblems, the front wall always moves to the right. The B&B process is repeated on the set of active subproblems.

When a generated subproblem becomes a feasible solution, its solution value is compared to the position of the back wall. If the position of the back wall is greater than the solution value, the back wall is set to this value; otherwise, the feasible solution is ignored. Successive expansions of subproblems cause the front and back walls to approach each other, and the process is terminated when the two walls meet.

In Sections II-A and II-B, the positions of the front and back walls for continuous valued B&B algorithms are derived. Section II-C considers the equivalent derivation for discrete valued B&B algorithms.

A. Position of the Front Wall

The solution of the following problem is desired: given the position of the front wall, what is the expected number of subproblems examined; or inversely, given the number of subproblems examined, what is the expected position of the front wall. The set of *examined subproblems* is the set of nonterminal nodes in the B&B tree, and consists of subproblems that have been processed. The following assumptions are made in the derivation.

A1) The set of differences between the lower bound of an expanded subproblem and the lower bound of its corresponding parent subproblem form a set of *independent, identically distributed* (or i.i.d.) random variables with the gamma density function

$$f_G(y;\alpha,\lambda) = \begin{cases} \dfrac{\lambda^\alpha}{\Gamma(\alpha)} y^{\alpha-1} e^{-\lambda y} & x > 0 \\ 0 & x \leq 0. \end{cases} \tag{8}$$

The density function is monotonic if $\alpha \leq 1$ and is unbounded near the origin when $\alpha < 1$. For $\alpha > 1$, the graph is bell-shaped, and, as $\alpha \to \infty$, the density function becomes normal [5]. A

gamma density function is chosen because it represents a very general class of density functions.

A2) Each parent subproblem is expanded into s (a constant) smaller subproblems. This assumption is valid for a wide class of combinatorial search problems. If the degree of branching is not constant, s will be assigned the average degree as an approximation.

Let $N(x)$ be the number of subproblems examined when the front wall is at position x and $E(N(x))$ be the expected value of $N(x)$. When a subproblem P_i is expanded into s child subproblems, $P_{i_1}, \cdots, P_{i_s}$, let $y_j = g(P_{i_j}) - g(P_i)$, $1 \leqslant j \leqslant s$. A subproblem at position x can be generated as the jth child of a parent at position $(x - y_j) > 0$. Hence, $E(N(x))$ can be written in the form of a renewal equation [25]:

$$E(N(x)) = 1 + \sum_{j=1}^{s} \left[\int_0^\infty E_1(N(x - y_j)) dF_G(y_j) \right]$$

$$= 1 + s \int_0^\infty E_1(N(x - y)) dF_G(y) \tag{9}$$

where

$$E_1(N(x - y)) = \begin{cases} E(N(x - y)) & \text{if } y < x \\ 0 & \text{if } y \geqslant x. \end{cases} \tag{10}$$

The above renewal equation cannot be solved analytically because it requires an incomplete gamma function as the distribution function. Since $f_G(y) \to 0$ as $y \to \infty$, the assumption that x is reasonably large implies that for any $y > x$, $f_G(y) \simeq 0$. This leads to the following approximate renewal equation:

$$E(\tilde{N}(x)) = 1 + s \int_0^\infty E(\tilde{N}(x - y)) dF_G(y). \tag{11}$$

To solve (11), a solution is guessed and is verified by substitution. Assuming that

$$E(\tilde{N}(x)) = ke^{mx} - \frac{1}{s - 1}$$

and substituting it into (11), we obtain an identity:

$$ke^{mx} - \frac{1}{s - 1} = 1 + s \int_0^\infty \left(ke^{m(x-y)} - \frac{1}{s - 1} \right) dF_G(y)$$

or

$$1 = s \int_0^\infty e^{-my} dF_G(y). \tag{12}$$

Substituting the density function in (8) into (12), m can be solved:

$$m = \lambda(s^{1/\alpha} - 1). \tag{13}$$

To solve for the constant k, we use the boundary condition $E(\tilde{N}(I_0)) = N_0$, where I_0 is the lower bound of the N_0th subproblem expanded. The value of N_0 should be chosen such that any initial transients in the density function in (8) are eliminated. Substituting x for I_0 in the assumed solution, we obtain

$$k = \frac{N_0(s - 1) + 1}{s - 1} e^{-mI_0}.$$

Therefore,

$$E(\tilde{N}(x)) = \frac{N_0(s - 1) + 1}{s - 1} e^{\lambda(s^{1/\alpha} - 1)(x - I_0)} - \frac{1}{s - 1}. \tag{14}$$

Note that the above equation is used as an approximation when s is not integral.

As with problems in general renewal theory, the derivation of the distribution function of $\tilde{N}(x)$ is difficult. The expected value of $\tilde{N}(x)$ will therefore be used in the calculation of the position of the back wall.

It is useful to know the total number of subproblems generated. All subproblems to the left of the front wall have been examined (nonterminal nodes), and all subproblems to the right of the front wall are active and have not been examined (terminal nodes). From (14), the expected total number of nodes in the B&B tree, when the front wall is at position x, is

$$E(\tilde{N}_T(x)) \simeq sE(\tilde{N}(x)) + 1. \tag{15}$$

The results derived in (14) also illustrate the effectiveness of approximate B&B algorithms. In general, suppose a feasible solution of value z_F is obtained when the approximate B&B process terminates, and let z_0 be the value of the optimal solution ($z_0 \leqslant z_F$). Let ϵ be the prescribed degree of accuracy [see (6)]. This implies that all subproblems with lower bounds greater than $z_F/(1 + \epsilon)$ are eliminated. From (14), the number of subproblems evaluated when the process terminates is

$$E\left(\tilde{N}\left(\frac{z_F}{1 + \epsilon}\right)\right) = \frac{N_0(s - 1) + 1}{s - 1} \cdot e^{\{\lambda[s^{1/\alpha} - 1][z_F/1 + e - I_0]\}} - \frac{1}{s - 1}. \tag{16}$$

The number of subproblems evaluated when the optimal solution is found is

$$E(\tilde{N}(z_0)) = \frac{N_0(s - 1) + 1}{s - 1} e^{\lambda[s^{1/\alpha} - 1](z_0 - I_0)} - \frac{1}{s - 1}. \tag{17}$$

The number of iterations saved is

$$E(\tilde{N}(z_0)) - E(\tilde{N}(z_F/(1 + \epsilon))).$$

If the final feasible solution obtained under approximation is very close to the optimal solution, say $z_F = z_0$, then the reduction in the number of iterations is proportional to

$$e^{[\lambda z_0 (s^{1/\alpha} - 1)]} - e^{[\lambda z_0 (s^{1/\alpha} - 1)/1 + e]}$$

which is exponential in ϵ. This simple derivation shows that a linear reduction in accuracy can result in an exponential reduction in the total number of iterations.

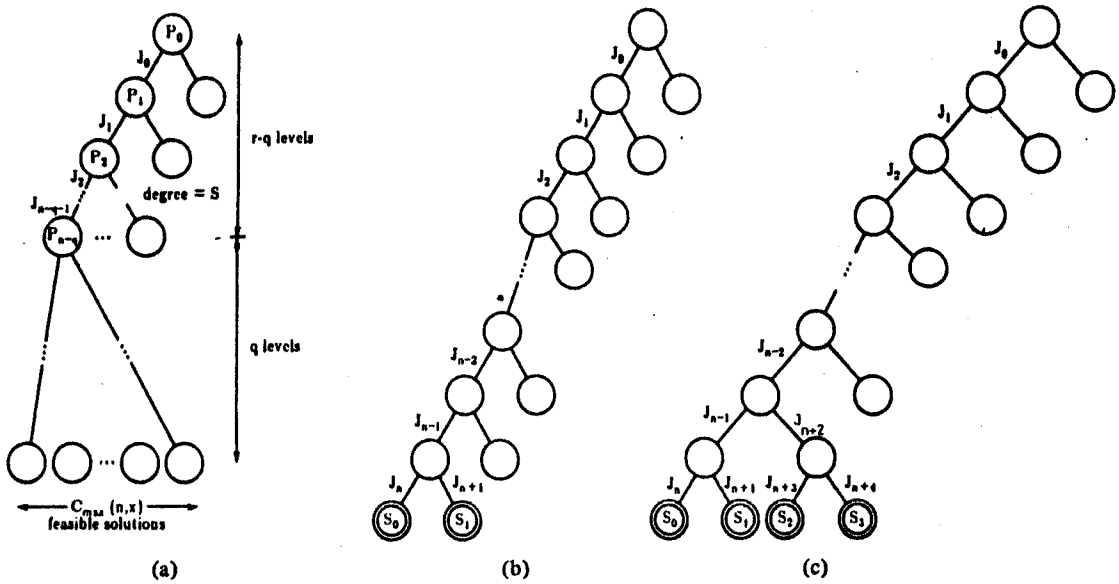


Fig. 3. Minimal branch-and-bound tree with (a) $C_{\max}(n, x)$, (b) two (with $s = 2$) and (c) four (with $s = 2$) solution nodes.

B. Position of the Back Wall

To determine the position of the back wall, the mechanism involved in generating a feasible solution must be understood. Let M be the number of input parameters. M can be the number of variables in an integer-programming problem, or the number of cities that a traveling salesman wishes to visit, or the number of nodes in the graph of a vertex-cover problem. Before a feasible solution is found, a chain of subproblem expansions leading to the feasible solution must be generated. The number of subproblems in a chain can be less than M (vertex-cover problem), equal to M (integer-programming problem), or greater than M (traveling-salesman problem). The chains may also have variable length. To simplify the evaluation of the position of the back wall, an additional assumption is made here.

A3) Every chain resulting in a feasible solution is made up of n (a constant integer) subproblem evaluations. Each chain has a length equal to the sum of n independent, gamma distributed random variables. When the chains have variable lengths, an average integral length is used for all chains as an approximation.

The number of chains due to $E(\tilde{N}_T(x))$ nodes in the B&B tree is $C(n, x)$. The maximum of $C(n, x)$, $C_{\max}(n, x)$, can be obtained by assuming that the B&B tree grows in a depth-first fashion as shown in Fig. 3(a). This tree can be thought of as consisting of two components, an upper skewed subtree portion rooted at P_0 and a lower full subtree portion rooted as P_{n-q}. The lower portion of the tree has a height of $q = \lceil \log_s C_{\max}(n, x) \rceil$, degree s at each nonterminal node, and $C_{\max}(n, x)$ terminal nodes. By Assumption A3, the length of a complete chain is n, so the upper portion of the tree has a height of $n - q$. $C_{\max}(n, x)$ is related to $E(\tilde{N}_T(x))$ as follows:

$$E(\tilde{N}_T(x)) = \sum_{j=0}^{q-1} \left\lceil \frac{C_{\max}(n,x)}{s^j} \right\rceil + s(n - q - 1) + 1. \qquad (18)$$

$C_{\max}(n, x)$ can be solved by first calculating its approximate value without the ceiling function in (18) and searching for the solution in the vicinity of the approximate value. The actual number of chains formed is, of course, less than $C_{\max}(n, x)$. The position of the back wall estimated using $C_{\max}(n, x)$ chains will therefore be a lower bound of the actual position. From (15) and (18), when the front wall is at x, the average number of active subproblems (or the average memory space required) is

$$E(\tilde{N}_{\text{active}}(x)) \simeq (s - 1)E(\tilde{N}(x)) + 1 - C_{\max}(n, x). \qquad (19)$$

Let $E[B(i)]$ be the expected position of the back wall when i feasible solutions have been generated. In the following analysis, the configuration of the tree in Fig. 3(a) is used, and thus s must be assumed to be a constant integer. Fig. 3(b) shows the B&B tree when two feasible solutions, S_0 and S_1, have been generated. Let $J_0, J_1, \cdots, J_{n+1}$ be the increase in lower bounds between the child and parent subproblems associated with the links in the figure. By Assumption A1, $J_0, J_1, \cdots, J_{n+1}$ are i.i.d. random variables with a density function $f_G(y; \alpha, \lambda)$ as defined in (8). Let l_0 be the lower bound of the initial problem (the root of the B&B tree) and J be a random variable with density $f_G(y; \alpha, \lambda)$. The two chains of subproblem expansions leading to feasible solutions S_0 and S_1 only differ in J_n and J_{n+1}. The back wall will be set to the smaller of the solution values of S_0 and S_1. Therefore,

$$E[B(2)] = l_0 + (n - 1)E[J] + E[\min(J_n, J_{n+1})]. \qquad (20)$$

Similarly, Fig. 3(c) depicts the B&B tree when four feasible solutions, S_0, S_1, S_2, and S_3, have been generated. The expected position of the back wall will be

$$E[B(4)] = l_0 + (n - 2)E[J] + E[\min(J_{n-1} + \min(J_n, J_{n+1}), J_{n+2} + \min(J_{n+3}, J_{n+4}))]. \qquad (21)$$

In general, when s^q feasible solutions have been generated,

$$E[B(s^q)] = l_0 + (n - q)E[J] + E[W_q] \qquad (22)$$

where $W_1, W_2, \cdots, W_q$ are random variables defined by the recursion $W_k = \min(J + W_{k-1}, \cdots, J + W_{k-1})$ with s terms in the minimization and the boundary condition that $W_1 = 0$. Let $F_{W_i}(y)$ be the distribution function of W_i. Then, $V_k = J + W_{k-1}$ has a density function $f_{V_k}(z)$ given by the convolution of the density functions of J and W_{k-1}, that is,

$$f_{V_k}(z) = \int_0^z f_G(z - x) dF_{W_{k-1}}(x). \tag{23}$$

W_k will have a distribution function given by

$$F_{W_k}(y) = 1 - \left[1 - \int_0^y f_{V_k}(z)\, dz\right]^s. \tag{24}$$

There is no closed form to (24) when the J's are gamma distributed; hence, the value of $E[B(i)]$ has to be solved numerically.

An approximation that requires less computation can be obtained by assuming that the chains are only dependent in $(n - q)$ links in the top portion of the tree, and all chains in the lower portion of the tree rooted at P_{n-q} are independent with a length equal to the sum of q independent gamma variables [Fig. 3(a)]. In this deriviation, the assumption on the integrality of s is unnecessary. Since the family of gamma densities is closed under convolution, the length of each subchain rooted at P_{n-q} is gamma distributed with a density function

$$f_q(y) = f_G(y; q\alpha, \lambda). \tag{25}$$

Assuming that all subchains rooted at P_{n-q} are independent, $E[W_q]$ is now approximated by the minimum value of all the subchains, denoted as $E[\tilde{W}_q]$. The distribution of the minimum of $C_{\max}(n, x)$ i.i.d. random variables with density given by (25) is

$$F_{\tilde{W}_q}(y) = 1 - [1 - F_q(y)]^{C_{\max}(n,x)}. \tag{26}$$

The expected position of the back wall is

$$E[B(C_{\max}(n,x))] = l_0 + (n - q)E[J] + \int_0^\infty y\, dF_{\tilde{W}_q}(y). \tag{27}$$

The solution to (27) has to be obtained numerically.

C. Front and Back Walls for Discrete-Valued B&B Algorithms

A derivation similar to that in Sections II-A and II-B can be carried out for B&B algorithms that are used to solve discrete valued problems such as vertex cover. Assumption A1 has to be modified, since the increase in lower bound when a subproblem is expanded is no longer governed by a continuous distribution. This increase is assumed to be an i.i.d., negative-binomial random variable with distribution

$$f(k; r, p) = \binom{r + k - 1}{k} p^r (1 - p)^k \qquad k = 0, 1, 2, \cdots. \tag{28}$$

r and k are parameters to be determined from the problem concerned. This distribution has been verified for vertex-cover problems.

Using the arguments and notations in Section II-A, an approximate renewal equation can be derived for the expected number of expanded subproblems when the front wall is at position x.

$$E(\tilde{N}(i)) = 1 + s \sum_{j=0}^{\infty} E(\tilde{N}(i - j)) \cdot f(k; r, p)$$
$$i = l_0, l_0 + 1, \cdots. \tag{29}$$

The solution to (29) is

$$E(\tilde{N}(i)) = \frac{N_0(s - 1) + 1}{s - 1} \left(\frac{q}{1 - ps^{1/r}}\right)^{(i - l_0)} - \frac{1}{s - 1}$$
$$i = l_0, l_0 + 1, \cdots. \tag{30}$$

Similarly, using the arguments and notations in Section II-B and assuming that s is integral, the expected position of the back wall after s^q feasible solutions have been generated is

$$E[B(s^q)] = l_0 + (n - q)E[J] + E[W_q] \tag{31}$$

where $E[W_q] = \sum_{i=0}^{\infty} i \cdot \Pr(W_q = i)$, and $W_k = \min[J + W_{k-1}, \cdots, J + W_{k-1}]$, $k = 2, 3, \cdots$, with s terms in the minimization, and $W_1 = 0$. Let $V_k = J + W_{k-1}$. The distribution of V_k is given by the convolution

$$\Pr(V_k = i) = \sum_{j=0}^{i} f(j; r, p) \cdot \Pr(W_{k-1} = i - j)$$

$\Pr(W_k = i)$ can be computed recursively as

$$\Pr(W_k = i) = \sum_{j=1}^{s} \binom{s}{j} [\Pr(V_k = i)]^j [\Pr(V_k > i)]^{(s-j)}$$
$$k = 2, 3, \cdots; i = l_0, l_0 + 1, \cdots. \tag{32}$$

A similar approximation can also be derived for the expected position of the back wall by assuming that the subchains in the lower portion of the B&B tree are independent. In this case, s, the degree of branching, is not restricted to integers. Since the family of negative binomial distributions is closed under convolution, $f_q(k) = f(k; qr, p)$. The expected position of the back wall is given by

$$E[B(C_{\max}(n,x))] = l_0 + (n - q)E[J] + \sum_{i=0}^{\infty} i \left[\left(\sum_{j=i}^{\infty} f_q(j)\right)^{C_{\max}(n,x)} - \left(\sum_{j=i+1}^{\infty} f_q(j)\right)^{C_{\max}(n,x)}\right]. \tag{33}$$

III. Verification of Analytical Model with Simulations

We have verified the analytical model with extensive simulations for the integer-programming, knapsack, and vertex-cover problems. The simulation programs were written in the C language and run on a DEC VAX 11/780 computer. We depict some of the results in Figs. 4–6.

In Fig. 4, the simulated and predicted positions of the front and back walls are plotted for two 17-variable, 17-constraint integer-programming problems that were randomly generated as follows. Referring to (7), C was a 1-by-17 vector, each element of which was a random integer between 0 and 10; A was

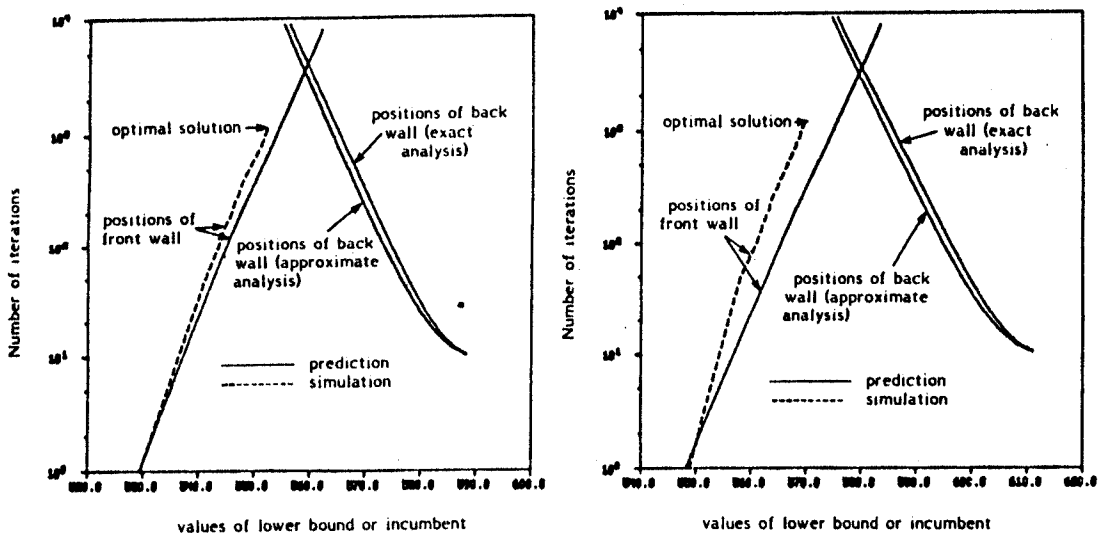


Fig. 4. Predicted and simulated positions of the front and back walls for two 17-variable, 17-constraint integer-programming problems.

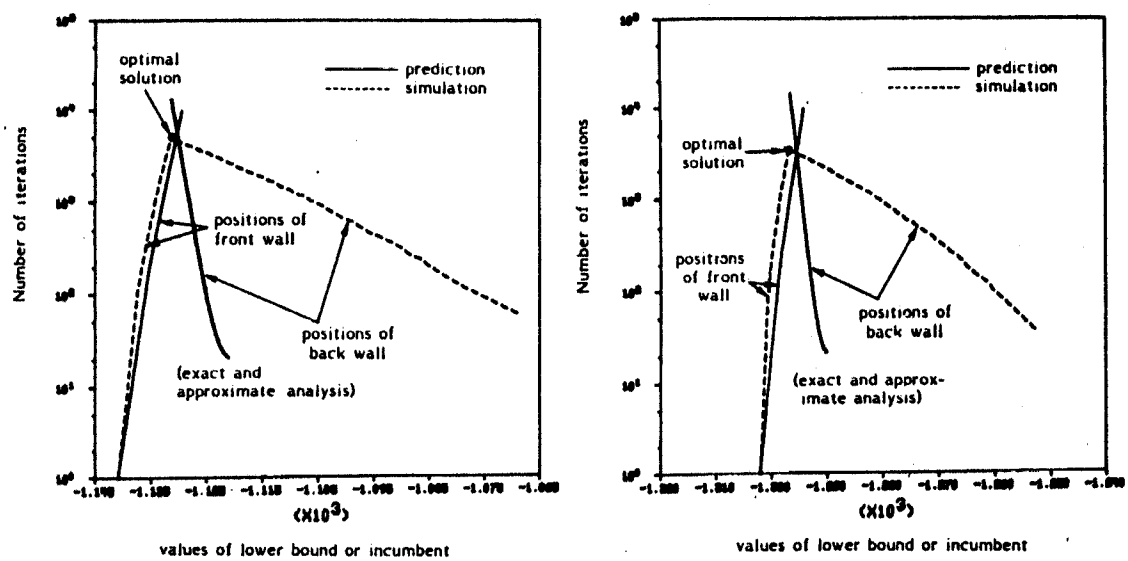
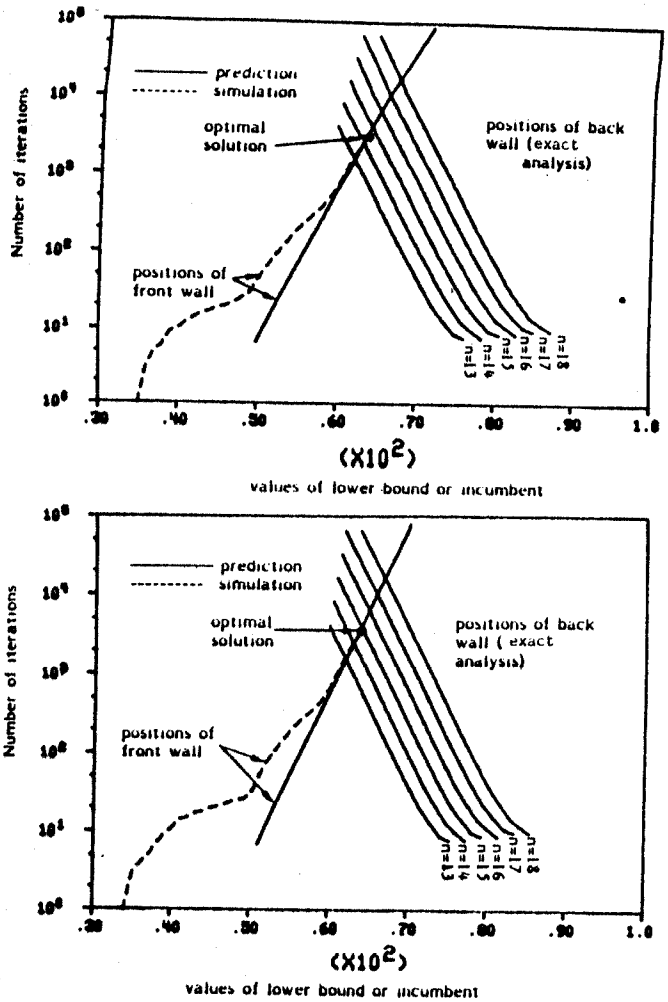


Fig. 5. Predicted and simulated positions of the front and back walls for two 30-object knapsack problems.

a 17-by-17 matrix, each element of which was an integer between -10 and 10; B was a 17-by-1 vector, each element of which was a random integer between -17 and 0. The values of α, λ, and s were estimated from the statistic of the problem. In the exact analysis of the back wall [see (22)], s was assumed to be 2. The values of N_0 and I_0 in (14) were taken to be 1 and I_0, respectively. Each problem took between 180 and 1040 s of CPU time to solve and required about 3 Mbytes of memory space. In contrast, the proposed analytical model required 124 kbytes of memory space and less than 0.9 s of CPU time. It is seen that the expected number of examined subproblems increases exponentially with the position of the front wall, and the position of the back wall approaches that of the front wall as the number of examined subproblems is increased.

In spite of the various assumptions made, the predicted positions of the front wall match to within 2 percent of the simulated positions. No results concerning the simulated positions of the back wall are plotted because no initial feasible solutions were generated in our runs, and the first feasible solution obtained usually became the optimal solution. Nonetheless, assuming a 2 percent error in the estimated positions of the back wall that are plotted in Fig. 4, the number of iterations at termination is predicted correctly. Due to the steepness of the curves and the exponential scale is used in the ordinate, the predicted number of iterations may lie in a range of several orders of magnitude. Other simulation results show similar behavior.

In Fig. 5, we have compared the results predicted by the model with the results obtained by simulations for two randomly generated, 30-object knapsack problems. Knapsack problems are maximization problems and are converted to minimization problems by negating the profits. Hence, the abscissas in Fig. 5 have negative values. The weight of each object is a random number between 1 and 100, and the profit is set to the value of the weight. The capacity of the knapsack is a random number between 0.5 and 0.8 of the total weight of the objects. The distribution function in (8) was approximated by an exponential distribution, and the values of d and s were collected from simulation statistics. The values of N_0 and I_0 were assumed to be 1 and I_0, respectively. Each prob-



ig. 6. Predicted and simulated positions of the front and back walls for two 80-node vertex-cover problems.

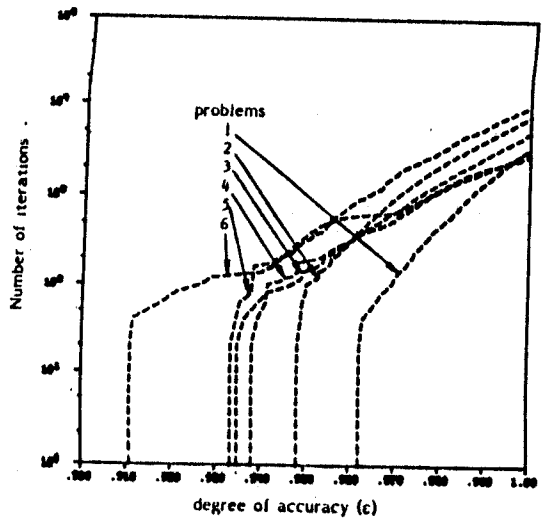


Fig. 7. Exponential reduction in the number of iterations with linear decrease in the accuracy of solutions for six 30-object knapsack problems.

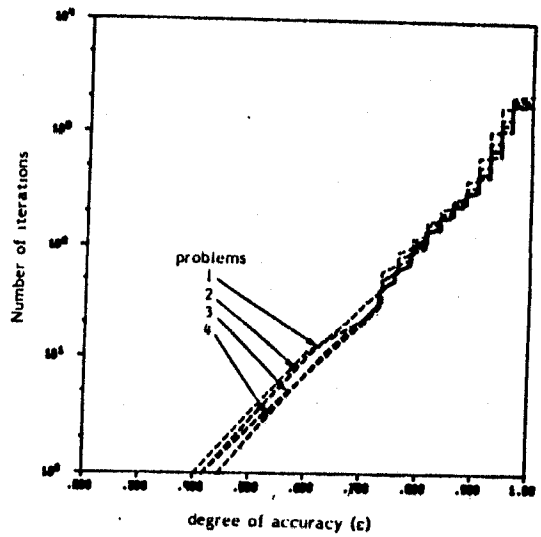


Fig. 8. Exponential reduction in the number of iterations with linear decrease in the accuracy of solutions for four 80-node vertex-cover problems.

m took between 120 and 14 000 s of CPU time to solve and equired about 0.1-0.3 Mbytes of memory space, while the nalytical model used 120 kbytes of memory space and approximately 2 s of CPU time. In the knapsack problems, many asible solutions were generated bedore the optimal solution as found, and the actual positions of the back wall can be otted. The estimated positions of the front wall lie within 5 percent of the actual positions. Although the estimated sitions of the back wall are much less than the actual positions, the number of iterations at termination is predicted rrectly to within a 0.5 percent error margin. Furthermore, e predicted positions of the back wall obtained by approximation [see (27)] are very close to the positions obtained by act analysis [(see (22)]. Other simulation results show simi- behavior.

Fig. 6 shows the results for two 80-node vertex-cover problems. The undirected graphs for the problems were generated domly with a probability 0.25 that an edge existed between air of nodes. The parameters s, r, and p were estimated m statistic collected on each problem. Due to some initial nsients in the problems, N_0 subproblems with $l_0 = l_0 + 3$ e used as the boundary condition in (30) and were not sidered in the density function in (28). Each problem solved in 170-550 s of CPU time and 1 Mbyte of memory e. The computation of the proposed stochastic model k 34 kbytes of memory space and less than 1 s of CPU time. The estimated positions of the front wall agree well with the actual positions after the initial transients are discounted. As noted in Section II-B, the length of a chain of subproblem expansions leading to a feasible solution is a variable for the vertex-cover problems. Furthermore, the distribution of the chain length is not known, as the first feasible solution generated is usually the optimal solution. In these figures, we have plotted a set of curves representing the positions of the back wall, assuming that all the chains have constant length. For the problems shown in Fig. 6, the best approximation is given by a length of 16. An analysis of the problem in Fig. 6(a) shows that the minimum chain length is 9 and the maximum chain length is 62. For the problem in Fig. 6(b), the chains have lengths between 5 and 65. Other simulation results show similar behavior.

We have also evaluated the effectiveness of approximate B&B algorithms for the knapsack and vertex-cover problems. As seen in Figs. 7 and 8, the number of required iterations decreases exponentially with a linear reduction in the accuracy of the approximate solution. A similar phenomenon has also been observed in our earlier simulations with the evaluation

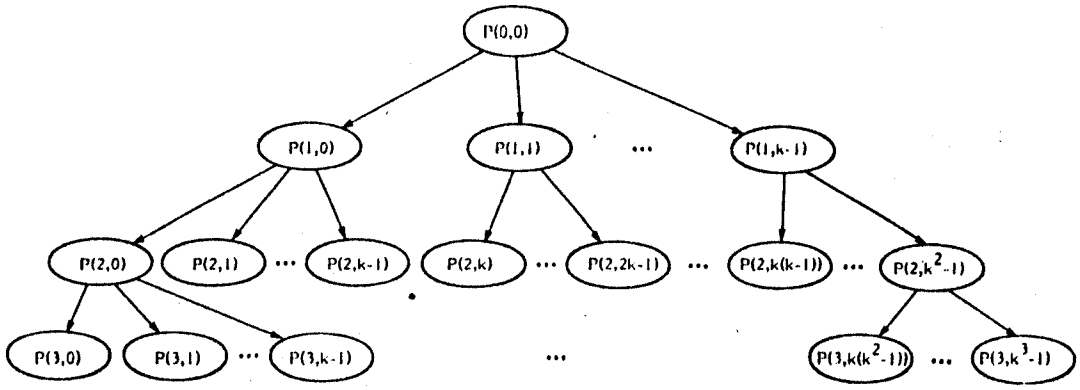


Fig. 9. A k-way state-space tree.

of vertex-cover problems by a parallel branch-and-bound algorithm [27].

IV. Comparison of Best-First and Depth-First Searches

In this section, we have derived and evaluated approximate expressions for the expected number of nodes expanded in best-first and depth-first searches of a given state-space tree. The model proposed in Section II is accurate in predicting the performance of a best-first search; however, it is difficult to extend this model to predict the performance of a depth-first search. An analytical model for a depth-first search would depend on the distribution of the accuracy of the lower-bound function in predicting the best solution value derivable from a given node. This distribution would be different from that of a best-first search , due to the different sequence of nodes examined. To compare the performance of search strategies for all combinations of distribution functions would be a nontrivial if not impossible problem. A mathematically tractable solution that assumes a constant distribution for the accuracy is presented here. The effects of the accuracy of the lower-bound function on the average performance of the A*-algorithm (best-first search) and informed backtracking (depth-first search) have been studied for decision problems [32]. Other models of the cutoff mechanism which do not explicitly consider the accuracy of the lower bounds have also been used to study the average performance of best-first [33] and depth-first [33], [34] searches.

Suppose that the state-space tree is a complete k-way tree of height h, as shown in Fig. 9. The root of the tree is at Level 0, and the leaves (solution nodes) are at Level h. Let $P(i, j)$, $0 \leq i \leq h$, $0 < j < \mu(i) - 1$, be the jth node in the ith level of the tree, where $\mu(i) = k^i$ is the number of nodes in Level i. Let S_j be the value of solution node $P(h, j)$, $0 \leq j \leq \mu(h) - 1$. It is assumed that all solution values are i.i.d. random variables with a distribution function $F_S(s)$ and a density function $f_S(s)$, where

$$f_S(s) = \begin{cases} \dfrac{\beta e^{\beta s}}{(e^{\beta b} - e^{\beta a})} & a \leq s \leq b \\ 0 & \text{otherwise.} \end{cases} \tag{34}$$

This density function is a truncated exponential distribution between a and b, and is chosen because of its ability to represent a wide variety of situations when β is varied. In particular, when $\beta \to -\infty$, the function is reduced to a delta function at a, which represents the situation in which all solutions are optimal. As β is increased, the standard deviation of the distribution first increases and then decreases to zero when $\beta \to \infty$, at which point the function is reduced to a delta function at b. An increase in the standard deviation of the distribution corresponds to a probabilistic decrease in the number of solutions with values in the vicinity of the value of the optimal solution.

As there are $\mu(h - i)$ solution nodes in a subtree rooted at $P(i, j)$, $z(i, j)$, the value of the best solution in this subtree, will be given by

$$z(i,j) = \min\{S_j \cdot \mu(h-i), \cdots, S_{(j+1)\cdot\mu(h-i)-1}\}. \tag{35}$$

z_0, the optimal solution, equals $z(0, 0)$. Since all the S_j's are i.i.d. random variables, it can be shown that $z(i, 0), \cdots, z(i, \mu(i) - 1)$ are also i.i.d. random variables with the following distribution function:

$$F_{z,i}(x) = \begin{cases} 0 & x < a \\ 1 - [1 - F_S(x)]^{\mu(h-i)} & a \leq x \leq b \\ 1 & x > b \end{cases}$$

$$= \begin{cases} 0 & x < a \\ 1 - \left(\dfrac{e^{\beta b} - e^{\beta x}}{e^{\beta b} - e^{\beta a}}\right)^{\mu(h-i)} & a \leq x \leq b \\ 1 & x > b. \end{cases} \tag{36}$$

Let $g(i, j)$ be the lower bound for $P(i, j)$, and $\delta(i, j)$ be the accuracy of the lower-bound function at $P(i, j)$. Then $g(i, j) = \delta(i, j) \cdot z(i, j)$. By definition, $\delta(h, j) = 1$ since $P(h, j)$ is a solution node. In this paper, we have only studied the case in which $\delta(i, j) = \delta$, $i < h$, and δ is a constant in the range $[a/b, 1]$. When δ is less than a/b, the elimination rule will be ineffective, and all nodes in the state-space tree must be expanded. Since $z(i, 0), \cdots, z(i, \mu(i) - 1)$ are i.i.d. random variables, $g(i, 0), \cdots, g(i, \mu(i) - 1)$ will also be i.i.d. random variables. For $i < h$, $g(i, 0), \cdots, g(i, \mu(i) - 1)$ will have a distribution function given by

$$F_{g,i}(x) = \begin{cases} 0 & x < \delta a \\ F_{z,i}(x/\delta) & \delta a \leq x \leq \delta b \\ 1 & a < \delta b < x. \end{cases} \tag{37}$$

Suppose that $N_B(i, j)$ denotes the number of nodes in the subtree rooted at $P(i, j)$ expanded during a best-first search,

assuming that $P(i, j)$ is expanded. A node $P(i, j)$ will be expanded in a best-first search if and only if $g(i, j) \leq z_0$ [see (2)]. $N_B(i, j)$, $0 \leq j \leq \mu(i) - 1$, can be expressed in the following recurrence equation:

$$N_B(i,j) = \begin{cases} 1 + \sum_{j_1=0}^{k-1} [\Pr\{g(i+1, kj+j_1) \leq z_0\} \\ \quad \cdot N_B(i+1, kj+j_1)] & i < h-1 \\ 1 & i = h-1. \end{cases} \quad (38)$$

Since $\Pr\{g(i, j) \leq z_0\}$ and $N_B(i, j)$ are independent, $E[N_B(i, j)]$, the expected value of $N_B(i, j)$, can be expressed as

$$E[N_B(i,j)] = 1 + \sum_{j_1=0}^{k-1} [E[\Pr\{g(i+1, kj+j_1) \leq z_0\}] \cdot E[N_B(i+1, kj+j_1)]] \quad i < h \quad (39)$$

$E[N_B(0, 0)]$ is the expected number of nodes that will be expanded during a best-first search, as the root is always expanded. Since $g(i, j)$, $j = 0, \cdots, \mu(i) - 1$, are i.i.d. random variables, $E[N_B(i, 0)] = \cdots = E[N_B(i, \mu(i) - 1]$, $0 \leq i \leq h$, and

$$E[N_B(i,0)] = 1 + k \cdot E[\Pr\{g(i+1, 0) \leq z_0\}] \cdot E[N_B(i+1, 0)] \quad i < h-1 \quad (40)$$

where

$$E[\Pr\{g(i,j) \leq z_0\}] = \int_a^b F_{g,i}(x) f_{z_0}(x)\, dx = \int_a^{\delta b} F_{z,i}(x/\delta) f_{z_0}(x)\, dx + \int_{\delta b}^{b} f_{z_0}(x)\, dx. \quad (41)$$

From (35), z_0 will have the following density function;

$$f_{z_0}(x) = \begin{cases} \mu(h) f_S(x) [1 - F_S(x)]^{\mu(h)-1} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} = \begin{cases} \dfrac{\mu(h)\beta e^{\beta x}(e^{\beta b} - e^{\beta x})^{\mu(h)-1}}{(e^{\beta b} - e^{\beta a})^{\mu(h)}} & a \leq x \leq b \\ 0 & \text{otherwise.} \end{cases} \quad (42)$$

Substituting (36) and (42) into (41) yields

$$E[\Pr\{g(i,j) \leq z_0\}] = 1 - \int_a^{\delta b} \cdot \frac{\mu(h)\beta e^{\beta x}(e^{\beta b} - e^{\beta x/\delta})^{\mu(h-i)}(e^{\beta b} - e^{\beta x})^{\mu(h)-1}}{(e^{\beta b} - e^{\beta a})^{\mu(h-i)+\mu(h)}}\, dx. \quad (43)$$

Similarly, let $N_D(i, j)$ be the number of nodes expanded in the subtree rooted at $P(i, j)$ during a depth-first search, assuming that $P(i, j)$ is expanded. Let $t(i, j)$ be the value of the incumbent when $P(i, j)$ is being considered for expansion. $P(i, j)$ will be expanded if and only if $g(i, j) \leq t(i, j)$. $N_D(i, j)$ can be written in the following recurrence equation:

$$N_D(i,j) = \begin{cases} 1 + \sum_{j_1=0}^{k-1} [\Pr\{g(i+1, kj+j_1) \leq t(i+1, kj+j_1)\} \\ \quad \cdot N_D(i+1, kj+j_1)] & i < h-1 \\ 1 & i = h-1. \end{cases} \quad (44)$$

Taking expectations of (44) results in

$$E[N_D(i,j)] = 1 + \sum_{j_1=0}^{k-1} [E[\Pr\{g(i+1, kj+j_1) \leq t(i+1, kj+j_1)\} \cdot E[N_D(i+1, kj+j_1)]] \quad i < h-1 \quad (45)$$

where

$$E[\Pr\{g(i,j) \leq t(i,j)\}] = \int_a^b F_{g,i}(x) f_{t,i}(x)\, dx = \int_a^{\delta b} F_{z,i}(x/\delta) f_{t,i}(x)\, dx + \int_{\delta b}^{f} f_{t,i}(x)\, dx. \quad (46)$$

The expected number of nodes expanded during a depth-first search will be $E[D(0, 0)]$. When $P(i, j)$ is being considered in a depth-first search, $j\mu(h - i)$ solution nodes have already been found. Thus,

$$t(i,j) = \min\{S_0, \cdots, S_{j\mu(h-i)-1}\}. \quad (47)$$

Hence, $t(i, j)$ has the following density function:

$$f_{t,i}(x) = \begin{cases} j\mu(h-i) f_S(x) [1 - F_S(x)]^{j\mu(h-i)-1} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} = \begin{cases} \dfrac{j\mu(h-i)\beta e^{\beta x}(e^{\beta b} - e^{\beta x})^{j\mu(h-i)-1}}{(e^{\beta b} - e^{\beta a})^{j\mu(h-i)}} & a \leq x \leq b \\ 0 & \text{otherwise.} \end{cases} \quad (48)$$

Substituting (36) and (48) into (46) yields

$$E[\Pr\{g(i,j) \leq t(i,j)\}] = 1 - \int_a^{\delta b} \cdot \frac{j\mu(h-i)\beta e^{\beta x}(e^{\beta b} - e^{\beta x/\delta})^{\mu(h-i)}(e^{\beta b} - e^{\beta x})^{j\mu(h-i)-1}}{(e^{\beta b} - e^{\beta a})^{(j+1)\mu(h-i)}}\, dx. \quad (49)$$

Equations (43) and (49) have no closed form and can only be expressed in the form of a recurrence. The solution of the recurrence would require $O(h \times k)$ recursions, which is extremely difficult to compute for nontrivial values of h and k. Instead, (43) and (49) were evaluated by numerical integration.

In Fig. 10, we depict the results of evaluating $E[N_B(0, 0)]$

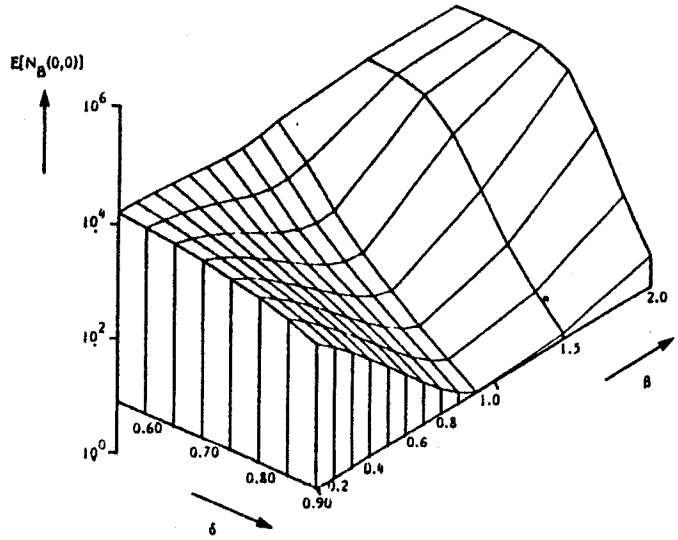


Fig. 10. $E[N_B(0, 0)]$ for a state-space tree with $k = 2$, $h = 15$, $a = 10$, $b = 20$.

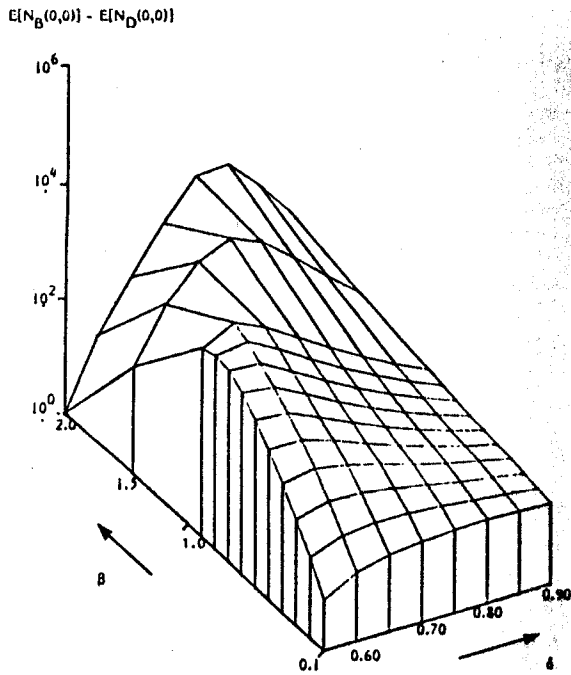


Fig. 11. Difference between $E[N_D(0, 0)]$ and $E[N_B(0, 0)]$ for a state-space tree with $k = 2$, $h = 15$, $a = 10$, $b = 20$.

for a tree with $k = 2$, $h = 15$, and various values of δ and β ($a = 10$, $b = 20$). The figure shows that the expected number of nodes expanded in a best-first search decreases as δ, the accuracy of the lower-bound function, increases. However, the amount and rate of decrease vary with β. For small values of β, there is only a small and gradual decrease. As β is increased, the amount and rate of decrease increase and then decrease. Recall that the standard deviation of the distribution increases and then decreases as β is increased. When the standard deviation is small, there are many solution nodes with values in the vicinity of the optimal solution. Therefore, the discriminatory power of the lower-bound function is weak, which results in a large number of expanded nodes. As the standard deviation of the distribution increases, the number of solution nodes in the vicinity of the optimal solution decreases, resulting in an increase in the discriminatory power of the lower-bound function and a smaller number of expanded nodes.

Fig. 11 shows the difference between $E[N_D(0, 0)]$ and $E[N_B(0, 0)]$ for the same binary tree and the same set of parameters. It can be seen that the expected number of nodes expanded in a best-first search is always less than that of a depth-first search. This difference first increases and then decreases as δ is increased. When δ is small (i.e., the lower-bound function is very inaccurate), depth-first and best-first searches perform equally badly and expand a substantial portion of the state-space tree. On the other hand, when δ is close to 1 (i.e., the lower-bound function is very accurate), depth-first search and best-first searches perform equally well and expand a small portion of the nodes in the state-space tree. This suggests that when the lower-bound function is either very good or very bad, there is little advantage of using a best-first search. Between these two extremes, a depth-first search always expands many more nodes than a best-first search. The regions for which the above behavior takes place depends on the value of β.

The behavior predicted for best-first and depth-first searches is illustrated by the 0/1 knapsack, vertex-cover, and integer-programming problems. The lower-bound functions for the 0/1 knapsack and vertex-cover problems are greedy algorithms and are very accurate in pruning unnecessary expansions. In contrast, the integer-programming problem uses a lower-bound function that is a linear program with the integrality constraints removed for all unassigned variables. This function is generally less accurate; hence, it is expected that the difference between the number of subproblems expanded in depth-first and best-first searches will be greater. This has been observed in our simulations with the 30-object 0/1 knapsack, 20-variable, 20-constraint integer-programming, and 80-node vertex-cover problems. Some of our results are shown in Table I. For the knapsack and vertex-cover problems, the number of subproblems expanded in a depth-first search was almost identical to that in a best-first search. For integer-programming problems, the ratio of the number of subproblems expanded in a depth-first search to that of a best-first search varies between 1.4 and 17.6. During the simulations, subproblems with identical lower bounds were evaluated in a consistent order to prevent anomalies between best-first and depth-first searches [31], [20]. Hence, a best-first search always expands a smaller number of nodes than a depth-first search.

TABLE I
NUMBER OF NODES EXPANDED IN BEST-FIRST AND DEPTH-FIRST SEARCHES FOR TEN 30-OBJECT 0/1 KNAPSACK, TEN 80-NODE VERTEX COVER, AND TEN 20-VARIABLE, 20-CONSTRAINT INTEGER-PROGRAMMING PROBLEMS

Problem	Number of Nodes Expanded					
	0/1-Knapsack		Vertex-Cover		Integer-Program	
	BFS	DFS	BFS	DFS	BFS	DFS
1	1701	1731	4367	6205	1544	2170
2	5475	5519	5837	7290	2113	5703
3	6394	6442	8050	9759	2241	5299
4	1406	1474	5350	7134	1244	9943
5	1103	1145	4462	5186	593	1177
6	15309	15403	8258	8277	1403	3056
7	808	860	12289	12463	356	2671
8	7137	7202	7559	7647	1461	17669
9	18265	18396	12808	21892	247	4347
10	14257	14337	14108	15524	3471	7862

V. Concluding Remarks and Implications

In this paper, we have studied the stochastic modeling of branch-and-bound algorithms under a best-first search. The model consists of two walls approaching each other. The front wall represents the value of the lower bound for the subproblem currently expanded. The back wall represents the minimum of all currently obtained feasible solutions. These two walls approach each other and eventually coincide at the termination of the process.

In deriving the positions of the walls, some simplifying assumptions were introduced to make the model mathematically tractable. In spite of these assumptions, the number of iterations at the termination of the process can be predicted accurately. This number is important, as it represents the minimum computational time to solve a problem by branch-and-bound methods. The analysis shows that the average number of iterations and the average maximum memory space required to obtain an optimal solution by a best-first search is exponential with a sublinear exponent, which coincides with the results obtained by Brown and Purdom on backtracking [30]. The exponential behavior in branch-and-bound algorithms is inherent unless the lower-bound function is very accurate in leading to the optimal solution. To further reduce the computational complexity, dominance relations similar to those used in dynamic programming can be applied [11]. The proposed model also demonstrates that a linear reduction in accuracy of the solution results in an exponential reduction in the number of iterations. The predicted performance is verified by simulations with integer-programming, knapsack, and vertex-cover problems.

We have also compared the number of nodes expanded under the best-first and depth-first searches as a function of the distribution of the values of the feasible solutions and the accuracy of the lower-bound function. Depth-first search has been found to have comparable computational complexity to best-first search when the lower-bound function is very accurate or very inaccurate; otherwise, best-first search is usually better. Results observed in our simulations agree with the evaluations of the analytical model.

The model has been applied to the design of an efficient branch-and-bound algorithm on a computer with a two-level memory hierarchy [28], [29].

References

[1] H. Berlinger, "The B* tree search algorithm: A best-first proof procedure," *Artificial Intell.*, vol. 12, pp. 23–40, 1979.

[2] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming.* New York: Academic, 1977.

[3] W. L. Eastman, "A solution to the traveling-salesman problem," presented at Amer. Summer Meet. Econometric Soc., Cambridge, MA, Aug. 1958.

[4] M. A. Efroymson and T. C. Ray, "A branch-and-bound algorithm for plant location," *Oper. Res.*, vol. 14, pp. 361–368, 1966.

[5] W. Feller, *An Introduction to Probability Theory and its Applications*, vol. II, 2nd ed. New York: Wiley, 1971.

[6] R. Garfinkel, "On partitioning the feasible set in a branch-and-bound algorithm for the asymmetric traveling-salesman problem," *Oper. Res.*, vol. 21, no. 1, pp. 340–342, 1973.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA: Freeman, 1979.

[8] A. M. Geoffrion and R. E. Marsten, "Integer programming algorithms: A framework and state-of-the-art survey," *Management Sci.*, vol. 18, pp. 465–491, May 1972.

[9] T. Ibaraki, "Computational efficiency of approximate branch-and-bound algorithms," *Math. Oper. Res.*, vol. 1, no. 3, pp. 287–298, 1976.

[10] —, "Theoretical comparisons of search strategies in branch-and-bound algorithms," *Int. J. Comput. Inform. Sci.*, vol. 5, no. 4, pp. 315–344, 1976.

[11] —, "The power of dominance relations in branch-and-bound algorithms," *J. ACM*, vol. 24, no. 2, pp. 264–279, 1977.

[12] —, "Depth-*n* search in branch-and-bound algorithms," *Int. J. Comput. Inform. Sci.*, vol. 7, no. 4, pp. 315–343, 1978.

[13] G. Ingargiola and J. Korsh, "A general algorithm for one-dimensional knapsack problems," *Oper. Res.*, vol. 25, no. 5, pp. 752–759, 1977.

[14] D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," *Artificial Intell.*, vol. 6, pp. 293–326, 1975.

[15] W. Kohler and K. Steiglitz, "Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems," *J. ACM*, vol. 21, no. 1, pp. 140–156, 1974.

[16] V. Kumar and L. Kanal, "A general branch-and-bound formulation for understanding and synthesizing and/or tree search procedures," *Artificial Intell.*, vol. 21, pp. 179–198, 1983.

[17] A. H. Land and A. Doig, "An automatic method for solving discrete programming problems," *Econometrica*, vol. 28, pp. 497–520, 1960.

[18] E. L. Lawler and D. W. Wood, "Branch-and-bound methods: A survey," *Oper. Res.*, vol. 14, pp. 699–719, 1966.

[19] J. Lenstra, "Sequencing by enumerative methods," Mathematisch Centrum, Amsterdam, Math. Cen. Tract 69, The Netherlands, 1976.

[20] G. J. Li and B. W. Wah, "Computational efficiency of parallel approximate branch-and-bound algorithms," in *Proc. Int. Conf. Parallel Processing*, Aug. 1984, pp. 473–480.

[21] A. Martelli and U. Montanari, "Optimizing decision trees through heuristically guided search," *Commun. ACM*, vol. 21, pp. 1025–1039, 1978.

[22] L. Mitten, "Branch-and-bound methods: General formulation and properties," *Oper. Res.*, vol. 18, pp. 24–34, 1970.

[23] N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence.* New York: McGraw-Hill, 1971.

[24] —, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.

[25] S. M. Ross, *Applied Probability Models with Optimization Applications.* San Francisco, CA; Holden-Day, 1970.

[26] G. C. Stockman and L. N. Kanal, "Problem reduction representation for the linguistic analysis of waveforms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-5, no. 3, pp. 287–298, 1983.

[27] B. W. Wah and Y. W. Ma, "MANIP–A multicomputer architecture for solving combinatorial extremum-search problems," *IEEE Trans. Comput.*, vol. C-33, pp. 377–390, May 1984.

[28] C. F. Yu and B. W. Wah, "Virtual-memory support for branch-and-bound algorithms," in *Proc. COMPSAC*, Chicago, IL, 1983, pp. 618–626.

[29] —, "Efficient branch-and-bound algorithms on a two-level memory system," in *Proc. COMPSAC*, Chicago, IL, 1984, pp. 504–514.

[30] C. A. Brown and P. W. Purdom, Jr., "An average time analysis of backtracking," *SIAM J. Comput.*, vol. 10, pp. 583–593, Aug. 1981.

[31] T. Ibaraki, "Theoretical comparisons of search strategies in branch and bound algorithms," *Int. J. Comput. Inform. Sci.*, vol. 5, no. 4, pp. 315–344, 1976.

[32] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Reading, MA: Addison-Wesley, 1984.

[33] D. R. Smith, "Random trees and the analysis of branch and bound procedures," *J. ACM*, vol. 31, no. 1, pp. 163–188, 1984.

[34] H. S. Stone, "The average complexity of depth-first search," IBM Res. Rep. RC 10717, 1984.

[35] B.-W. Wah, G.-J. Li, and C.-F. Yu, "Multiprocessing of combinatorial search problems," *IEEE Computer*, vol. 18, no. 6, pp. 93–108, June 1985.

Benjamin W. Wah (S'74-M'79) received the B.S. and M.S. degrees in electrical engineering and computer science from Columbia University, New York, NY, in 1974 and 1975, respectively, and the M.S. degree in computer science and the Ph.D. degree in engineering from the Univer-

sity of California, Berkeley, in 1976 and 1979, respectively.

He was on the faculty in the School of Electrical Engineering, Purdue University, between 1979 and 1985. He is now an Associate Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois, Urbana-Champaign. His current research interests include parallel computer architectures, distributed databases, artificial intelligence, and theory of algorithms.

Dr. Wah has been a Distinguished Visitor of the IEEE Computer Society since 1983.

Chee Fen Yu (S'82) received the B.E. (Hons.) degree in electrical engineering from the University of Malaya, Kuala Lumpur, Malaysia, in 1980, and the M.S. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1983.

He is currently working towards the Ph.D. degree in electrical engineering at Purdue University. His current research interests include computer architecture and artificial intelligence.

Environmental Testing Techniques for Software Certification

MYRON S. KARASIK, MEMBER, IEEE

Abstract—The problems of developing software requirements and quality assurance techniques have basically dealt with an environment where a single organization acts as the designer, developer, and user of the software product. Since the mid-1970's, however, there has been a great increase in the use of "packaged" software products designed and developed by one organization for use in a variety of other organizations. The great profusion of products has resulted in many products being peddled for generic applications (accounting, manufacturing, etc.) which are of questionable quality and/or "fit" to a given organization's environment. This paper describes some techniques that are being used to certify software produced by third parties and how to determine if the "fit" is there. Current quality assurance techniques deal with the "correctness" of a program as compared to its specifications [2], [4], [7], [8], [12]. The real issue for a purchaser of software is whether the software is "correct" for its environment.

Index Terms—Environmental testing, logical pathway, quality assurance, system testing, test generator, test plans.

I. Background

ALL programs by their nature have only the following command classes:

Direct Action: arithmetic, go to, read, write, assignment.

Conditional Choice: comparative, logical and recursive (IF-THEN; AND, OR, NOT; DO WHILE; CASE)

The database structure of a software package defines the distinguishable data elements and their characteristics. Quality assurance system testing as performed by a developer or third party determines whether the actual database structure matches the database design and whether the program's commands are correct [3], [9] (by producing outputs that represent the designed specifications for valid combinations of database elements or properly changing the database when appropriate activity transactions are processed).

Manuscript received November 7, 1983.

The author is with OASAS Limited, Chicago, IL 60601.

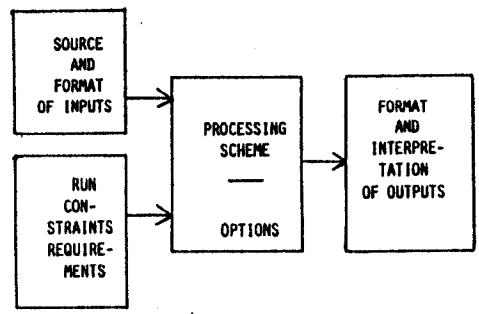


Fig. 1. Information requirements for system test.

Quality assurance procedures are based on the specifications document and use (or design) of a test generator [5], [10] (manual or automated) to test each logical pathway in the system [6] (each branch of which is generated by conditional logic). Fig. 1 gives the information requirements and Fig. 2 is an example of a logical pathway.

This set may be large but such exhaustive testing is possible, having been done at Bell Labs Business Information Systems Area [1]. The methodology employed is described in Appendix I.

II. Enviromental Testing Concepts

The performance of environmental testing basically must start from one of two points:

- a detailed specifications document prepared by the potential user giving the database and processing requirements which can then be compared to a certified detailed capabilities document of the potentially acquired software package;

0098-5589/85/0900-0934$01.00