Random Trees and the Analysis of Branch and Bound Procedures

DOUGLAS R. SMITH

Office of Naval Research, Arlington, Virginia

Abstract. Branch and bound procedures are the most efficient known means for solving many NP-hard problems A special class of branch and bound procedures called *relaxation-guided* procedures is presented. While for some branch and bound procedures a worst-case complexity bound is known, the average case complexity is usually unknown, despite the fact that it may give more useful information about the performance of the algorithm. A random process which generates labeled trees is introduced as a model of the kind of trees that a relaxation-guided procedure generates over random instances of a problem Results concerning the expected time and space complexity of searching these random trees are derived with respect to several search strategies. The best-bound search strategy is shown to be optimal in both time and space. These results are illustrated by data from random traveling salesman instances Evidence is presented that the asymmetric traveling salesman problem can be solved exactly in time $O(n^3 \ln(n))$ on the average.

Categories and Subject Descriptors: F.2.2 **[Analysis of Algorithms and Problem Complexity]**: Nonnumerical Algorithms and Problems, G.2.1 **[Discrete Mathematics]**: Combinatorics; G.1.6 **[Numerical Analysis]**. Optimization

General Terms: Algorithms, Theory

Additional Key Words and Phrases· Discrete optimization, branch and bound procedures, random processes, average case analysis of algorithms, search strategies, traveling salesman problem

1. *Introduction*

A discrete minimization problem Π is a triple $\Pi = \langle S, D, f\rangle$ where S is a discrete set of objects called feasible solutions, D is a set of input instances, and $f: S \times D \to Z^+$ (positive integers) is a cost function. An *instance* of Π has the form $\langle S', d, f\rangle$ where $S' \subseteq S$ and $d \in D$. $\langle S_1, d, f\rangle$ is a *subinstance* of $\langle S_0, d, f\rangle$ if $S_1 \subseteq S_0$. When no confusion can arise we will identify an instance by S', its set of feasible solutions. An *optimal solution* to an instance S' is an object $x \in S'$ which has minimal cost; that is, for all $y \in S'$ $f(x, d) \le f(y, d)$. The goal is to find an optimal solution to a given instance of Π. The basic branch and bound procedure works as follows. An instance of a problem Π is analyzed, and if the minimal-cost object is not easily extracted, then the instance is decomposed into subinstances and a lower bound is computed on the cost of the minimal-cost object in each subinstance. Those subinstances whose bound exceeds the cost of some known (perhaps nonoptimal) solution can be discarded since they cannot contain an optimal solution. The remaining subinstances are repeatedly analyzed, decomposed, and bounded until an object is found whose cost does not exceed the bound

This work was supported in part by NSF grant MC74-14445-AO1.

Author's address: Office of Naval Research, Code 433, Information Sciences Division, Room 631, 800 N. Quincy Street, Arlington, VA. 22217.

on any subinstance, hence that object is an optimal solution. The search process generated by a branch and bound procedure is often represented by a tree in which instances are represented by nodes, and the decomposition of instance S_i into subinstances is represented by arcs from the node representing S_i to each subinstance.

Branch and bound seems to have emerged as the principal method for solving discrete minimization problems which are classified as NP-hard [6]. Theoretical treatments of branch and bound procedures may be found in [2], [14–16], [22], [25], [31], and [33]. Just a few of the applications of the branch and bound method include integer programming [8], flow shop and job shop sequencing [17], traveling salesman problems [3, 4], knapsack problems [9, 13], puzzles and other cognitive tasks [32], optimal decision tress [30], and pattern recognition [18].

In this paper we focus on a class of branch and bound procedures which we call relaxation guided. A *relaxation* of a discrete minimization problem $\Pi = \langle S, D, f\rangle$ is a problem $\Pi' = \langle T, D, g\rangle$ where $S \subseteq T$, and for all $x \in S$ and $d \in D$ $f(x, d) = g(x, d)$. Each instance S_i of Π corresponds to a unique instance T_i of Π' such that $S \cap T_i = S_i$. T_i is called the *relaxed instance* with respect to S_i. A *relaxation-guided* procedure makes use of a fast algorithm for solving Π'. If an optimal solution z to a relaxed instance T_i is also feasible ($z \in S_i$), then z is also an optimal solution to instance S_i. If z is not feasible, then it is used to decompose T_i into subinstances in such a way that z is precluded from further consideration. A few of the problems for which relaxation-guided algorithms have been devised are the symmetric and asymmetric traveling salesman problems [4, 11, 12, 37], the integer linear programming [8], quadratic assignment [24], set covering [27], and knapsack problems [13].

The complexity of an algorithm has usually been measured by its worst-case behavior over all instances of a problem (i.e., an upper bound on its performance). The obvious problem with such a measure is that it gives little information about the usual or average performance of the algorithm. For example, Klee and Minty [20] construct some examples which cause the simplex algorithm for solving linear programs to run in exponential time, yet its usual performance is so good that it is one of the most widely used computer algorithms. It is especially true of branch and bound procedures—which can have widely varying behaviors over the set of instances of a problem—that the average case complexity gives more useful information than a worst-case measure about the performance of the algorithm.

In this paper we use a random process similar to a branching process [10] in order to model the kinds of trees generated by a branch and bound procedure. The model enables us to derive several results on the expected time and space complexity of branch and bound procedures under best-bound-first and depth-first search strategies. With these results a model of a subtour-elimination algorithm for the traveling salesman is constructed, and empirical performance data are compared with values predicted by the model.

Section 2 discusses the branch and bound procedure. In Section 3 our model of branch and bound search trees is introduced and properties of the model trees are derived. Sections 4 and 5 develop results on the complexity of best-bound-first, depth-first, and general search strategies. Also in Section 5, the expected time complexity of a depth-first search is studied as a function of the depth of the first solution found in the search tree. A subtour-elimination algorithm for the traveling salesman problem is modeled in Section 6, and evidence is provided that it has an expected running time of $O(n^3\ln(n))$.

2. *Branch and Bound Procedures*

A branch and bound procedure has three major components. A *branching function* B is a rule determining if and how a given instance is to be decomposed into subinstances. In a relaxation-guided branch and bound procedure B uses an algorithm A which finds solutions to the relaxed instances; that is, $A: 2^T \to T$ such that for any $T_i \subseteq T$ and for all $y \in T_i\, f(A(T_i), d) \leq f(y, d)$. Formally B is a mapping from subinstances of T to a collection of subinstances of T ($B: 2^T \to 2^{2^T}$) such that

(1) if $A(T_i) \in S$ or T_i is a singleton, then $B(T_i)$ is the empty set (i.e., no instances are produced);
(2) otherwise, if $A(T_i) \notin S$, then $B(T_i) = \{T_{i1}, T_{i2}, \ldots, T_{ik}\}$ where
  (a) $\bigcup_{j=1}^{k} T_{ij} \subset T_i$,
  (b) $\bigcup_{j=1}^{k} S_{ij} = S_i$, and where $T_i \cap S = S_i$ and $T_{ij} \cap S = S_{ij}$,
  (c) $A(T_i) \notin \bigcup_{j=1}^{k} T_{ij}$.

The second component of a branch and bound procedure is a *lower bound function* (LB) which maps subinstances of T into nonnegative integers. For any instance $\langle T_i, d, f\rangle$ define

$$\mathrm{LB}(T_i, d) = \begin{cases} \infty & \text{if } T_i \text{ is the empty set} \\ f(A(T_i), d) & \text{otherwise.} \end{cases}$$

The following properties are satisfied by this definition of LB:

(1) $\mathrm{LB}(T_i, d) \leq f(s, d)$ for all $s \in S_i = T_i \cap S$, since $A(T_i)$ is a least-cost object in T_i;
(2) $\mathrm{LB}(T_i, d) \leq \mathrm{LB}(T_{ij}, d)$ if $T_{ij} \in B(T_i)$, since $T_{ij} \subseteq T_i$;
(3) if $A(T_i) \in S$, then $A(T_i)$ is the minimal-cost object in $S_i = T_i \cap S$.

When $A(T_i) \in S$, the branching function does not decompose the instance, precisely because an optimal solution of the instance has been found. In practice there may be ways to further improve the lower bound beyond the simple approach given above. See, for example, [12] and [37].

The lower bound function is used to eliminate from consideration those subinstances of T which can be shown not to contain an optimal solution to the original instance. If it is known that the optimal solution has a cost of at most c_1, then any instance T_i for which $\mathrm{LB}(T_i, d) > c_1$ cannot yield an optimal solution. An instance is said to have been *explored* if it has been terminated by this bounding test or if the branching function has been applied to it. An instance which has been generated by the branching function, but not yet explored at some point during a computation, is said to be *active*.

The third component of a branch and bound procedure is a *search strategy*, which is a rule for choosing to which of the currently active instances of S the branching rule should be applied. For conceptual simplicity and uniformity of notation, a search strategy will be realized here by a heuristic function $h: 2^T \to$ *priority*, where *priority* is a linearly ordered set which depends on the particular search strategy. Of those instances waiting to be explored via the branching rule we choose that instance T_i with the smallest heuristic value $h(T_i)$.

A relaxation-guided branch and bound procedure for finding a single least-cost object is given below in a Pascal-like language. The principal data structure is a *priority queue* which stores instances with an associated priority given by the heuristic function h. The queue is accessible only by the following three operators: NONEMPTY, which returns true if and only if the priority queue is nonempty;

SELECT, which removes and returns the instance in the priority queue with smallest heuristic value; and INSERT, which inserts an instance into the priority queue with its associated heuristic value. Efficient algorithms for priority queues are discussed in [1].

The code assumes the user-defined data types *instance* (representation of relaxed problem instances); *solutiontype* (representation of T), *priority* (representation of priorities); and *priority queue* (representations for priority queues). In addition, the following generic functions are employed:

A: *instance* $\rightarrow$ *solutiontype* (A finds solutions to relaxed instances);
LB: *instance* $\rightarrow$ *integer* (LB computes the lower bound function);
FEASIBLE: *solutiontype* $\rightarrow$ *boolean* (FEASIBLE distinguishes elements of S from elements of $T - S$).

```
function BB(N: instance, UB: integer): solutiontype;
var
  i, j: integer;
  solution: solutiontype;
  PQ: priority queue;
begin
 1 if FEASIBLE(A(N)) then return(A(N));
 2 INSERT(N, PQ, h(N));
 3 while NONEMPTY(PQ) do         /* while there are active instances ... */
 4    begin N := SELECT(PQ);     /* find instance with least heuristic value */
 5      if LB(N) < UB            /* and explore it ... */
 6        then if FEASIBLE(A(N)) /* if relaxed solution is feasible ... */
            then begin           /* then save it */
 7             UB := LB(N); solution := A(N)
            end
          else begin             /* otherwise apply the branching rule ... */
 8          Apply branching rule to N generating subinstances N_1, N_2, ..., N_k;
 9          for i := 1 to k do   /* and store the subinstances */
10             INSERT(N_i, PQ, h(N_i))
          end
      end;
11 return(solution)
end
```

The procedure BB is typically invoked with T and ∞ as arguments where ∞ is an upper bound on the cost of any object in T. The variable UB serves to record the cost of the least-cost feasible object currently known during the search process. An obvious improvement of BB is to check that $\mathrm{LB}(N_i) < \mathrm{UB}$ in statement 10, before the instance N_i is inserted in the priority queue. While such a test will improve the performance of BB somewhat in practice, we omit it here for the sake of simplifying our analysis of the behavior of BB. Its inclusion would not affect our order of magnitude results on the time complexity of branch and bound search, but would have the effect of lowering the space complexity somewhat. In practice several other enhancements of the pruning power of BB may be added to this code, but they depend on the special structure of a particular problem. A *dominance relation* [15, 16, 22] is a relation on subinstances of T such that if T_i dominates T_j, then T_j cannot contain a better solution than T_i, so T_j can be eliminated from further consideration. An *equivalence test* [16] is based on an equivalence relation over subinstances of T. In many applications the branching structure generated by BB is a graph; a well-chosen equivalence test between nodes can eliminate much redundant search.

The *best-bound-first* (bbf) search strategy [5, 25] chooses to apply the branching rule to that active instance with the smallest lower bound. This strategy is realized by the heuristic function

$$h(T_i) = \mathrm{LB}(T_i)$$

where LB is the lower-bound function. The relation $\leq_h$ is just the usual relation $\leq$ on the integers. In practice a priority queue is indeed the appropriate data structure for implementing a best-bound-first search. Also, in practice ties will be broken in favor of deeper nodes in the search tree since, in general, it represents a more tightly constrained instance and may be closer to producing a relaxed solution that is feasible.

The *ordered-depth-first* (odf) search strategy applies the branching rule to the least cost of the most recently generated instances, and may be realized by

$$h(T_i) = (d(T_i), \mathrm{LB}(T_i))$$

where $d(T_i)$ = depth of the instance T_i in the tree generated by **BB** and the range of h is the set of ordered pairs of integers. The *generation-order-depth-first* (godf) search strategy applies the branching rule to the first generated of the most recently generated instances and can be realized by

$$h(T_i) = (d(T_i), i)$$

for the ith generated instance. Again, h produces an ordered pair. For both of these heuristic functions we define $(a, b) \leq_h (c, d)$ if and only if $a > c$ or ($a = c$ and $b \leq d$).

The *breadth-first* search strategies are another well-known class of search strategies which are, however, rarely used in relaxation-guided branch and bound procedures. Under a breadth-first search strategy all instances generated at depth $i \geq 0$ are explored before the subinstances they generate at depth $i + 1$.

As an example of a relaxation-guided branch and bound procedure we will consider a subtour-elimination algorithm for solving the traveling salesman problem (TSP). An n-city TSP can be described as follows: $\mathrm{TSP}_n = \langle C_n, D, f\rangle$ where C_n is the set of cyclic permutations of $\{1, 2, 3, \ldots, n\}$, D is the set of positive integral $n \times n$ matrices, and

$$f(\pi, d) = \sum_{i=1}^{n} d(i, \pi_i) \qquad \text{for all} \quad \pi \in C_n, \qquad d \in D.$$

The well-known assignment problem is used as the relaxation of the traveling salesman problem. An assignment problem of order n may be described by $\mathrm{ASSIGN}_n = \langle S_n, D, f\rangle$ where D and f are as defined above, and S_n is the set of permutations of $\{1, 2, \ldots, n\}$. The assignment problem is solvable in $O(n^3)$ time for an initial instance and $O(n^2)$ for subsequent modified instances [4] and [26].

Subtour-elimination algorithms differ mainly in their choice of branching rule. The following branching rule was proposed by Shapiro [34]: Given cost matrix d, solve the assignment problem with respect to d. If the least-cost solution, π, is cyclic, then we have extracted the least-cost cyclic permutation over the feasible set of d, so there is no need to branch. If π is noncyclic, then pick one of its subcycles, say the smallest, and let this cycle be denoted $(i_1, i_2, \ldots, i_k)$. In an optimal cost cyclic permutation at least one of the nodes in this cycle must be directed outside the cycle, since the subcycle cannot be a part of a cyclic permutation. The instance is decomposed as follows: In the jth subinstance the node i_j is forced to connect to a node not in the cycle $(i_1, i_2, \ldots, i_k)$ by setting the matrix entries $d_{i_j i_m} = \infty$ for $1 \leq m \leq k$. Throughout this paper we use the symbol ∞ to denote a number which is sufficiently large in context to be effectively infinite. Variations on this branching rule are given in [3], [7], and [37].

3. *A Model of Branch and Bound Search Trees*

3.1 Random Instances and Random Trees. It was noted previously that the branch and bound process generates a tree structure. In this section we use this abstraction to define a probabilistic model of the kind of tree structures that BB generates over the instances of a discrete minimization problem. Within this model we can derive various results concerning the expected time and space requirements of BB under various search strategies.

The collection of subinstances of T that are inserted in the priority queue during the execution of BB is called the *search tree*; the *time complexity* of a branch and bound search will be measured by the size of the search tree. The *space complexity* will be measured by the maximum number of instances in the queue at any time during the search. The time and space complexities of a given search by BB under search strategy h, given an initial bound of b, will sometimes be denoted by the variables $N_T(b)$ and $N_S(b)$, respectively. When appropriately defined the expected value of N_T and N_S will be denoted $E_h(N_T(b))$ and $E_h(N_S(b))$. When the initial bound is ∞, we will simply write N_T and N_S in place of $N_T(\infty)$ and $N_S(\infty)$. This definition of time complexity does not include the amount of time spent executing the branching rule or inserting nodes in the queue. We assume that these times are relatively independent of the choice of branching function, so they should factor out of the comparison of the different search strategies, leaving the size of the search tree as the essential measure of performance.

The question of interest is how to model the behavior of BB on a random instance of a problem, apart from the details of the problem. That is, which features of a branch and bound search are relevant to branch and bound and which are problem dependent? First, by the action of the branching rule a tree structure is generated so that BB is a tree searching algorithm. Second, the lower bound function of BB associates a number with each node in this tree. The search strategy does not affect the tree per se, but only the order in which the algorithm examines the tree. So a tree with costs associated with each node is another way of representing the domain of BB. In this setting the goal of BB is to find the least-cost leaf of the tree. These considerations are formalized in the following definition. An *arc-labeled tree* is a tree $t = (N, A, C)$ where N is a set of nodes, A is a set of arcs, and $C: A \rightarrow Z^+$ (positive integers) is a cost function on the arcs of the tree. (For example, see Figure 1.) In an arc-labeled tree the cost of a node is defined to be the sum of the costs on the arcs on the path from the root to the node. The cost of the root is zero.

The next step is to map the notion of a random instance of given size into the arc-labeled tree domain. A probability function is assigned to the class of arc-labeled trees which should somehow correspond with a probability measure on the instances of a discrete minimization problem. Our model of this mapping is to regard the generation of a tree as a random process in which each application of the branching rule is replaced by an independent random experiment where the outcome is the number of sons that a node has. In a similar manner, the assignment of a cost to a node is treated as the outcome of a different independent random

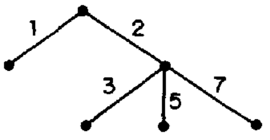


FIG. 1. An arc-labeled tree.

experiment. Formally, let P and Q be probability mass functions. The mean of P will be written $\bar{P}$. It is assumed that P and Q satisfy the following properties:

(1) $P(0) > 0$ (a node is terminal with nonzero probability),
(2) $Q(0) = 0$ (an arc has cost zero with probability zero).

Let RANDOM(F) be a random function which returns k with probability $F(k)$. The following procedure generates a random arc-labeled tree.

(1) Let a root node exist. The root is unsprouted.
(2) Select an unsprouted node n (according to some search strategy) and sprout it as follows: Let n have RANDOM(P) sons. For each arc from n to its sons, label the arc with cost RANDOM(Q).
(3) Repeat step 2 until all nodes have been sprouted.

This dynamic means of defining a random arc-labeled tree is easily implemented on a machine for experimental purposes. This process is related to the well-known branching process [10], which has applications to population growth, nuclear fission reactions, and particle cascades.

We need to define a probability function on the set of arc-labeled trees. This can be accomplished as follows. The generation of a tree is viewed as a sequence of trials, where each sprouting of step 2 in the above procedure is a trial. Let j denote the number of sons generated in a random trial and let $c_1, c_2, \ldots, c_j$ denote the arc costs assigned to the arcs. The probability of the outcome of a trial then is $P(j)Q(c_1)Q(c_2) \cdots Q(c_j)$. Clearly, if we sum over all possible outcomes of a trial, the probabilities sum to 1,

$$\sum_{n=0}^{\infty} P(n) \sum_{c_1=1}^{\infty} Q(c_1) \cdots \sum_{c_j=1}^{\infty} Q(c_j) = 1.$$

We can formulate the probability of a tree generated by this process as follows. Consider the probabilities of the outcomes of the trials during the generation of a tree in a sequence $\langle g_0, g_1, g_2, \cdots \rangle$, where g_i is the probability of the particular outcome of the ith trial. Let us call the product $g_0 g_1 \cdots g_i$ the ith partial probability of the randomly generated tree. The probability of a randomly generated tree then is the limit as i goes to infinity of the ith partial probability. For example, the probability of the arc-labeled tree in Figure 1 is $P(2)Q(1)Q(2) \cdot P(0)P(3)Q(3)Q(5)Q(7)P(0)P(0)P(0)$.

Let sons(n) denote the number of sons of the node n. The arc-labeled tree $t = (N, A, C)$ is in the class of (P, Q)-*random trees* if and only if (sons(n)) > 0 for all $n \in N$ and $Q(C(a)) > 0$ for all $a \in A$. We will omit the prefix (P, Q) and call t a random tree when P and Q are clear from context.

The key simplifying assumption in this model is the independence of each application of the branching rule and the independence of each assignment of arc costs. The independence assumptions will not hold exactly in branch and bound applications. Many branch and bound algorithms are not relaxation guided and find their solutions at a fixed depth k. In these algorithms the opportunity of finding an optimal solution at an intermediate depth $d < k$ is precluded and, thus, they are not well modeled by a depth-independent P. In general the branching and arc cost probabilities are dependent on many factors, some of them problem dependent. A more sophisticated problem-independent model might include the depth of a node as a parameter in P and Q. Alternatively, one might include as parameters the number of siblings of a node and the cost of the arc to its parent, thus modeling the branching rule and arc cost assignment by the state transition probabilities of a Markov process.

The key strength of the model is that it allows a tractable and general analysis of the expected performance of branch and bound under several search strategies. We claim that the model is useful in analyzing relaxation-guided branch and bound procedures and we provide an example in Section 6. More broadly, the independence assumptions may play a role in understanding the asymptotic properties of general branch and bound procedures. In many branch and bound applications each branching corresponds to the imposition of some constraints on a small number of variables in a parent instance. If the average number of variables affected by a decomposition is asymptotically negligible in comparison to the total number of variables in the problem, then we might expect that as the problem size increases the statistical properties of the child instance become more and more like the statistical properties of the parent instance. In the limit we may have statistical independence. In addition, for sufficiently large trees the branch and bound process examines only the topmost part of the full tree, which may have much more uniform properties than the tree as a whole. We present evidence in Section 6 that the trees produced by the branching function of a subtour-elimination algorithm from the traveling salesman problem do have a degree of uniformity in the probability of various branching factors. This uniformity can be exploited in the asymptotic analysis and and derivation of bounds on the expected running time of subtour-elimination algorithms.

3.2 Properties of a Class of (P, Q)-random Trees. Before studying the behavior of BB it will be useful to develop expressions for some important properties of a class of random trees. For example, what is the expected path length of a randomly picked path in a random tree? The probability that a node is a leaf is $P(0)$ and the probability that a node has some sons is $1 - P(0)$. A branch of length k then has probability $(1 - P(0))^k P(0)$, a geometric distribution. The expected path length is

$$\sum_{k=0}^{\infty} k(1 - P(0))^k P(0) = (1 - P(0))/P(0). \tag{3.1}$$

A more difficult question concerns the distribution of least-cost leaves over the class of random trees. Let $\text{lcl}(t)$ denote the cost of a least-cost leaf in an arc-labeled tree t. Let $\hat{O}(i)$ denote the probability that $\text{lcl}(t) = i$ in a random tree t. $\hat{O}$ is defined on the nonnegative integers, since the cost of any leaf in a random tree is a nonnegative integer by definition. A recurrence relation for $\hat{O}$ can be formulated by equating two expressions for the probability that $\text{lcl}(t) > i$ in a random tree t. First note that no arcs can have a cost of zero, so the only way that a tree can have a least-cost leaf of cost zero is if the root is terminal, thus $\hat{O}(0) = P(0)$. One expression for the probability that $\text{lcl}(t) > i$ is

$$1 - \sum_{k=0}^{i} \hat{O}(k). \tag{3.2}$$

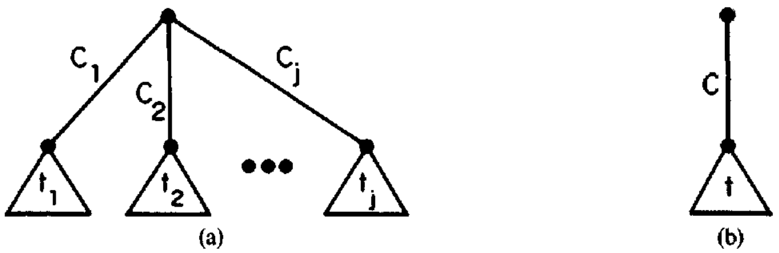


FIG. 2. (a) A treetop. (b) A branch.

Next consider the treetop shown in Figure 2a. The subtrees $t_1, t_2, \ldots, t_j$ are themselves random trees. The probability that $\mathrm{lcl}(t_k) > i$ where t_k is the kth subtree plus the arc from the root as in Figure 2b is

$$1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c). \tag{3.3}$$

This expression sums over all combinations of arc costs c and costs of least-cost leaves within t (letting s denote the least-cost leaf of the combined arc and subtree, $s - c$ is the cost of the least-cost leaf of the subtree) for which the sum is not greater than i. Since this expression applies independently to each of any number of branches, the probability that the treetop of Figure 2a has j branches and $\mathrm{lcl}(t) > i$ is

$$P(j)\left[1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}.$$

For $i > 0$ the probability that $\mathrm{lcl}(t) > i$ in a random tree t is

$$\sum_{j=1}^{\infty} P(j)\left[1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}. \tag{3.4}$$

The case $j = 0$ is not included in this expression because then $\mathrm{lcl}(t) = 0$. Finally, expressions (3.2) and (3.4) can be equated:

$$1 - \sum_{k=0}^{i} \hat{O}(k) = \sum_{j=1}^{\infty} P(j)\left[1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}. \tag{3.5}$$

This is a recurrence relation since $\hat{O}(i)$ appears on the left but only the values $\hat{O}(0)$, $\hat{O}(1), \ldots, \hat{O}(i-1)$ appear on the right for $i \geq 1$. In Appendix A this recurrence relation is broken down into simpler recurrence relations in order to speed up the computation of $\hat{O}$. Except for special P and Q, this recurrence relation seems to have no general analytic solution. Extrapolation of $\hat{O}$ based on uniformly distributed P and Q suggests that $\hat{O}(n)$ is asymptotic to d^n for some constant d that depends on P and Q. Figure 3 shows some of $\hat{O}$ for the class of (P_{10}, Q_{100})-trees where $P_{10}(k) = 1/11$ if $0 \leq k \leq 10$ and $P_{10}(k) = 0$ otherwise, and $Q_{100}(c) = 1/100$ if $1 \leq c \leq 100$ and 0 otherwise. Other properties of random trees, such as the probability that the shallowest least-cost leaf in a random tree occurs at depth k, are given in [36].

4. *Best-Bound-First and Heuristic Search Strategies*

Let $n_0, n_1, n_2 \ldots$ denote the sequence of nodes explored in a tree by BB under a heuristic search strategy h. n_i is called the *first-found leaf* if no node n_j with $j < i$ is

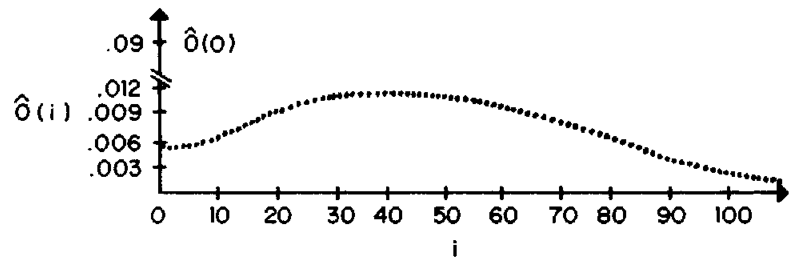


FIG. 3. $\hat{O}(i)$ for (P_{10}, Q_{100})-trees.

a leaf. The first-found leaf corresponds to the first feasible solution explored during a branch and bound search and has the effect of changing UB so that pruning can take place of subsequently explored nodes whose cost exceeds UB.

PROPOSITION 4.1. *If n_k is the first-found leaf in a best-bound-first search of an arc-labeled tree t, then n_k is the least-cost leaf in t.*

PROOF. A best-bound-first search explores the nodes of a tree t in order of nondecreasing cost. Let s be any leaf. At the moment that the first leaf node s^* is found, some ancestor of s is on the priority queue and has cost no less than the cost of s^*. Since costs are nondecreasing as we proceed down any path in an arc-labeled tree, s must have cost no less than the cost of s^*. Thus, s^* has minimal cost. Q.E.D.

As a consequence of Proposition 4.1 we can slightly modify a best-bound-first implementation of BB so that it terminates as soon as a leaf (representing a feasible solution) is found. With this proposition we can derive an expression for the expected time and space complexity of BB under a best-bound-first search strategy.

THEOREM 4.1. *Over a class of (P, Q)-random trees the expected time and space complexities of BB under the best-bound-first search strategy is given by*

$$E_{bbf}(N_T) = 1 + \frac{\bar{P}}{P(0)}$$
$$E_{bbf}(N_S) = 2 + \frac{\bar{P} - 1}{P(0)}.$$

PROOF. Let x_i be a random variable whose value is the number of children of n_i, the ith explored node. The number of nodes inserted in the priority queue prior to the exploration of the first-found leaf is a random sum of the form

$$1 + x_0 + x_1 + \cdots + x_{k-1} \tag{4.1}$$

where n_k is the first-found leaf. The first term in (4.1) counts the insertion of the root into the priority queue. The remaining terms count the children of each explored node prior to the first-found leaf. If n_k is the first-found leaf, then the space complexity is measured by

$$\max_{0 \le i < k} \left\{ 1 + \sum_{j=0}^{i} (x_j - 1) \right\} = 1 + \sum_{j=0}^{k-1} (x_j - 1) \tag{4.2}$$

since $x_j \geq 1$. The first term counts the insertion of the root into the priority queue and the jth term, $(x_j - 1)$, in (4.2) counts the children added to the queue by n_j and the removal of n_j itself from the queue. Each sum, (4.1) and (4.2), is a random sum of independently and identically distributed random variables whose mean is simply the product of the expected number of terms and the expected value of a term. The probability that the first-found leaf is n_k is $(1 - P(0))^k P(0)$ which has mean

$$\sum_{k=0}^{\infty} k(1 - P(0))^k P(0) = \frac{1 - P(0)}{P(0)}. \tag{4.3}$$

The probability that x_j has value m given that $m > 0$ is $P(m)/(1 - P(0))$ which has mean

$$\sum_{m=1}^{\infty} \frac{mP(m)}{1 - P(0)} = \frac{\bar{P}}{1 - P(0)}. \tag{4.4}$$

Taking the product of (4.3) and (4.4) and adding one for the root gives us the expected value of (4.1)

$$1 + \frac{\bar{P}}{1 - P(0)} \frac{1 - P(0)}{P(0)} = 1 + \frac{\bar{P}}{P(0)}. \tag{4.5}$$

The expected value of the random variable $(x_j - 1)$ in (4.2) is

$$\frac{\sum_{m=1}^{\infty} (m - 1)P(m)}{1 - P(0)} = \frac{\sum_{m=1}^{\infty} mP(m) - \sum_{m=1}^{\infty} P(m)}{1 - P(0)}$$
$$= \frac{\bar{P} - (1 - P(0))}{1 - P(0)}. \tag{4.6}$$

Adding 1 for the root to the product of (4.3) and (4.6) gives us the expected value of (4.2)

$$1 + \frac{\bar{P} - (1 - P(0))}{1 - P(0)} \frac{1 - P(0)}{P(0)} = 2 + \frac{(\bar{P} - 1)}{P(0)}. \qquad \text{Q.E.D.} \tag{4.7}$$

COROLLARY 4.1. *For any search strategy h, the expected time and space complexities of BB on a random tree have the following bounds:*

$$E_h(N_T) \geq 1 + \frac{\bar{P}}{P(0)}$$
$$E_h(N_S) \geq 2 + \frac{(\bar{P} - 1)}{P(0)}.$$

PROOF. Under any search strategy h, BB cannot terminate until at least one leaf has been removed from the priority queue. Thus N_T is bounded below by the number of nodes inserted in the priority queue prior to the exploration of the first-found leaf. This quantity is just (4.1) which has mean (4.5). Similarly N_S is bounded below by the number of nodes in the priority queue just prior to the removal of the first-found leaf. This quantity is measured by (4.2) and has mean (4.7). Q.E.D.

Theorem 4.1 and its corollary provide us with several interesting conclusions. First, the best-bound-first search strategy is optimal in the sense that its average-case performance is no worse than that of any other search strategy. Second, Theorem 4.1 indicates that in the construction of an efficient relaxation-guided branch and bound procedure we should seek to minimize the expected branching factor and maximize the chance that an optimal solution to a relaxed instance is also feasible. Third, only the relative ordering of the costs of the nodes in a tree are important to a best-bound-first search, not the particular costs, thus (4.5) and (4.7) are independent of Q.

The absence of Q in (4.5) means that the performance change in BB due to replacing one lower-bound function with another cannot be modeled by simply varying Q. Instead, new P and Q functions are needed which more closely model the search trees produced by BB (as opposed to modeling the full trees determined by the branching and lower-bound functions). If the search trees produced by BB, using a new lower-bound function, are smaller, it is because the first-found leaf is found sooner in the search. Thus $\hat{P}(0)$ is higher, and/or fewer nodes are generated by the branching function, thus $\bar{P}$ is lower. So it is possible to capture the effect of different lower-bound functions if one is willing to adopt the more sophisticated approach of attempting to model the search trees produced by BB.

5. *Depth-First Search Strategies*

5.1 EXPECTED TIME COMPLEXITY. The choice of which node to explore next by a depth-first search strategy is made between the sons of the most recently explored node (if any), otherwise the sons of the next most recently explored node, and so on. Let $ET(b)$ abbreviate $E(N_T(b))$. ET will be subscripted with godf and odf when necessary to distinguish between generation-order and ordered depth-first search strategies.

An expression for ET can be formulated naturally as a recurrence relation. Define the *effective bound* on a node N to be the value of the expression UB − LB(N) at the moment that N is explored by BB. The concept of effective bound enables us to treat each node as the root of a search tree with an initial value of UB which is just the effective bound. Suppose that BB is searching a tree with the structure shown in Figure 2a where each subtree $t_1, t_2, \ldots, t_j$ may be regarded as a random tree. Let b_0 be a finite initial value of UB (the effective bound on the root) and let b_i denote the effective bound on the root of t_i for $1 \leq i \leq j$. Then the expected size of the search tree for a random tree with this structure is

$$ET(b_0) = 1 + ET(b_1) + ET(b_2) + \cdots + ET(b_j).$$

Each of the bounds b_i for $1 \leq i \leq j$ is less than b_0, indicating that a recurrence relation may be set up. The next problem concerns the probability of a given bound occurring at a given node. Consider the tree in Figure 2a. Given an initial bound of b_0, the bound on the subtree t_1 is $b_0 - c_1$ so BB is expected to search $ET(b_0 - c_1)$ nodes in t_1. lcl(t_1) = m with probability $\hat{Q}(m)$ since t_1 is a random tree. The same holds for the other subtrees. Suppose that lcl(t_1) = m_1. If UB = b_0 initially, then after searching the first subtree of the tree of Figure 2a UB = $\min\{b_0, c_1 + m_1\}$. The effective bound on the root of the subtree t_2 is $\min\{b_0, c_1 + m_1\} - c_2$. Continuing this reasoning, one finds that the effective bound on the ith subtree t_i is

$$b_i = \min\{b_0, c_1 + m_1, c_2 + m_2, \ldots, c_{i-1} + m_{i-1}\} - c_i \tag{5.1}$$

where $m_1, m_2, \ldots, m_{i-1}$ denote the cost of the least-cost leaves of the subtrees t_1, $t_2, \ldots, t_{i-1}$, respectively. ET evaluated with expression (5.1) yields the expected number of nodes explored in t_i.

Let the functions $W_{\text{odf}}(b, i)$ and $W_{\text{godf}}(b, i)$ denote the expected size of the search tree of t_i when the root is given an initial bound of b for an ordered-depth-first search and a generation-order-depth-first search, respectively. An expression for

$W_{\text{godf}}(b, i)$ can be found by enumerating all possible combinations of variables in (5.1).

$$W_{\text{godf}}(b, i) = \sum_{c_1=1}^{\infty} \cdots \sum_{c_i=1}^{\infty} \sum_{m_1=0}^{\infty} \cdots \sum_{m_{i-1}=0}^{\infty} Q(c_1) \cdots Q(c_i)\hat{Q}(m_1) \cdots \hat{Q}(m_{i-1}) \cdot ET_{\text{godf}}(\min\{b, c_1 + m_1, \ldots, c_{i-1} + m_{i-1}\} - c_i). \tag{5.2}$$

A tree with a structure as in Figure 2a will have expected size

$$1 + \sum_{i=1}^{j} W_{\text{godf}}(b, i).$$

This expression summed over all j (number of sons of the root) gives an expression for $ET_{\text{godf}}(b)$:

$$\begin{aligned} ET_{\text{godf}}(b) &= \sum_{j=0}^{\infty} P(j) \left(1 + \sum_{i=1}^{j} W_{\text{godf}}(b, i)\right) \\ &= 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} W_{\text{godf}}(b, i). \end{aligned} \tag{5.3}$$

As stated, (5.3) is computationally intractable; however, it can be refined to a more computable form as given in Appendix A.

The order of examining the subtrees of Figure 2a by an ordered-depth-first search is treated as follows. In an arbitrary tree with this structure the arc costs $c_1, c_2, \ldots, c_j$ are unordered. By rearranging the tree the arc costs can be brought into sorted order. Note, though, that a given ordered sequence $c_1 \le c_2 \le \cdots \le c_i$ may result from the sorting of many distinct sequences. The appropriate combinatorial question is how many unique arrangements $R_i(c_1, c_2, \ldots, c_i)$ of this sequence there are. There are $i!$ nonunique arrangements, but repetitions must be accounted for. If k of the i values have the same value $c_{j+1} = c_{j+2} = \cdots = c_{j+k}$, then there is a repetition factor of $k!$ due to this relation. In general

$$R_i(c_1, c_2, \ldots, c_i) = \frac{i!}{r_1! r_2! \cdots r_k!}$$

where

$$c_1 = \cdots = c_{r_1} < c_{r_1+1} = \cdots = c_{r_1+r_2} < \cdots < c_{r_1+r_2+\cdots+r_{k-1}} = \cdots = c_{r_1+r_2+\cdots+r_k}.$$

and $r_1 + r_2 + \cdots + r_k = i$ (i.e., there are r_1 variables with the same value, r_2 variables with the same value, and so on).

Again, by enumerating all possible ordered sequences $c_1, \ldots, c_i$ and $m_1, \ldots, m_{i-1}$ of the variables in (5.1), an expression for $W_{\text{odf}}(b, i)$ can be found.

$$\begin{aligned} W_{\text{odf}}(b, i) = \sum_{c_1=1}^{\infty} \sum_{c_2=c_1}^{\infty} \cdots \sum_{c_{i-1}=c_i}^{\infty} Q(c_1)Q(c_2) \cdots Q(c_i) \\ \cdot R_i(c_1, c_2, \ldots, c_i) \sum_{m_1=0}^{\infty} \cdots \sum_{m_{i-1}=0}^{\infty} \hat{Q}(m_1) \cdots \hat{Q}(m_{i-1}) \\ \cdot ET_{\text{odf}}(\min\{b, c_1 + m_1, \ldots, c_{i-1} + m_{i-1}\} - c_i). \end{aligned} \tag{5.4}$$

A tree with a structure as in Figure 2a will have expected size

$$1 + \sum_{i=1}^{j} W_{\text{odf}}(b, i).$$

This expression summed over all j (number of sons of the root) gives an expression for $ET_{\text{odf}}(b)$:

$$ET_{\text{odf}}(b) = \sum_{j=0}^{\infty} P(j)\left(1 + \sum_{i=1}^{j} W_{\text{odf}}(b, i)\right)$$
$$= 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} W_{\text{odf}}(b, i). \tag{5.5}$$

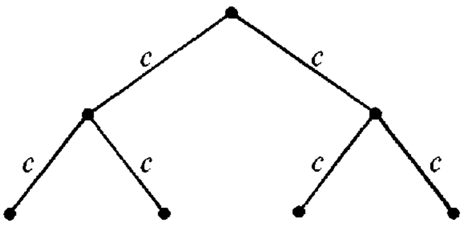
The following theorems assert that over most classes of random trees a best-bound-first search has strictly smaller expected time and space complexity than a depth-first search.

THEOREM 5.1. *Let df be any depth-first search strategy. If* $P(0) + P(1) < 1$ *then* $E_{df}(N_T) > 1 + \bar{P}/P(0)$.

PROOF. If t is a random tree on which a df search is performed, let N_T^t denote the corresponding value of the random variable N_T, and let α_t denote the number of nodes inserted in the priority queue just prior to the removal of the first-found leaf. By the argument of Corollary 4.1, $N_T^t \geq \alpha_t$ for all t. We will construct a random tree t' which has nonzero probability, and such that $N_T^{t'} > \alpha_{t'}$ then, using Theorem 4.1,

$$E_{\text{df}}(N_T) = \Pr(t')N_T^{t'} + \sum_{t \neq t'} \Pr(t)N_T^t > \Pr(t')\alpha_{t'} + \sum_{t \neq t'} \Pr(t)\alpha_t = 1 + \frac{\bar{P}}{P(0)}.$$

From $\sum_k P(k) = 1$ and $P(0) + P(1) < 1$ it follows that $P(i) > 0$ for some $i > 1$. Assume for simplicity that $i = 2$. The following construction can be easily generalized. Let c be a positive integer such that $Q(c) > 0$. t' has nonzero probability $P(0)^4P(2)^3Q(c)^6$, $\alpha_{t'} = 5$, and $N_T^{t'} = 7$.

$t' =$ 

Q.E.D.

THEOREM 5.2. *Let df be any depth-first search strategy. If either* (1) $P(0) + P(i) < 1$ *for all* $i > 0$, *or* (2) $P(0) + P(1) < 1$ *and* $Q(c) < 1$ *for all* c, *then*

$$E_{df}(N_S) > 2 + \frac{(\bar{P} - 1)}{P(0)}.$$

PROOF. The proof of this theorem is directly analogous to that for Theorem 5.1.

5.2 TIME COMPLEXITY AS A FUNCTION OF THE DEPTH OF THE FIRST-FOUND LEAF. The depth of the first-found leaf in a depth-first search has a strong effect on the performance of the search. Intuitively, if this depth is great, then the procedure will spend much of its time exploring nodes deep in the tree before returning to shallower levels where the least-cost leaf may lie. It might be conjectured that the size of a search tree tends to grow exponentially in the depth of the first leaf which it finds. To the contrary, in our model the size of a depth-first search tree is essentially linear in the depth of the first-found leaf.

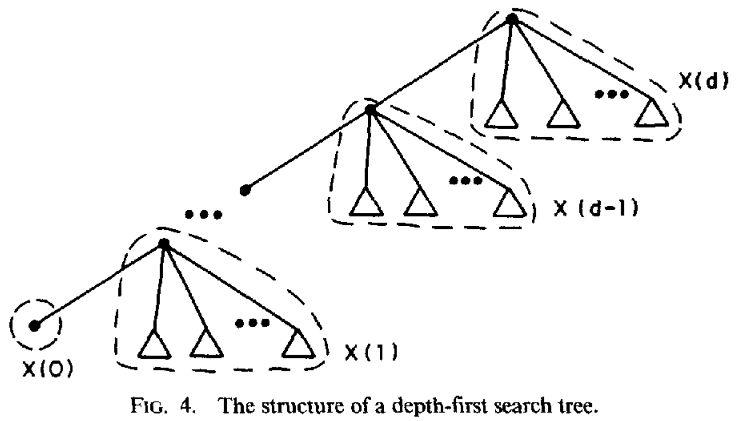


FIG. 4. The structure of a depth-first search tree.

Let $S(d)$ be the expected number of nodes searched in a random tree, given that the first solution occurs at depth d.

THEOREM 5.3. *For classes of random trees in which* $\lim_{b\to\infty} ET(b)$, *denoted* $ET(\infty)$, *exists,* $S(d)$ *exists and is bounded above and below by linear functions of* d.

PROOF. Let $X(d)$ denote the expected number of nodes in the search tree, except those in the first explored subtree, given that the first solution is found at depth d. $X(0)$ is defined to be 1. (See Figure 4.) From the definitions we have

$$S(d) = 1 + \sum_{k=1}^{d} X(k). \tag{5.6}$$

Since $X(k) \geq 1$, we have

$$S(d) = 1 + \sum_{k=1}^{d} X(k) \geq 1 + \sum_{k=1}^{d} 1 = 1 + d.$$

It is shown in [36] that there is a constant α such that for all d $X(d) \leq \alpha \leq ET(\infty)$. It follows that

$$S(d) \leq 1 + \sum_{k=1}^{d} \alpha = 1 + \alpha d. \qquad \text{Q.E.D.}$$

Theorem 5.3 can be interpreted as follows: A depth-first search tree can be decomposed along the path from the root to the first-found leaf into groups of subtrees whose expected size $X(i)$ is asymptotically constant (the ith group consists of the 2nd, 3rd, ..., jth subtrees below the ith node on the path from the root to the first-found solution).

All that is required for the proof of Theorem 5.3 is the existence of an upper bound α on the sequence $\{X(d)\}$. The formal proof of the existence of α depends on the independence assumptions of the model and on the existence of $ET(\infty)$. We conjecture, however, that such a bound exists even for branch and bound algorithms which are not well modeled by our assumptions. Intuitively, $ET(\infty)$ may provide a suitable bound. It would not be difficult to try to observe the linear behavior predicted by Theorem 5.3 in branch and bound applications, but we have not attempted to do so.

6. *An Application to the Traveling Salesman Problem*

A model of a particular branch and bound algorithm is an appropriate choice of P and Q functions parameterized by the problem size. By analysis of the initial relaxed instance for subtour-elimination algorithms for the traveling salesman problem, we can derive limiting expressions for P_n where $P_n(k)$ is the probability that a random instance of the n-city TSP is split into k subinstances. As described in Section 2 this algorithm makes use of a relaxation of the requirement that feasible objects be cyclic permutations, and the initial relaxed instance corresponds to the set S_n of permutations of n objects.

We assume that our set of input instances D is a class of cost matrices whose entries are independently and identically distributed random variables. The problem is to find the least-cost permutation with respect to a given cost matrix. The set S_n is a symmetric set in the sense that for any given pair $\pi_1, \pi_2 \in S_n$ there is a relabeling (automorphism) of the permutations of S_n such that π_1 is mapped into π_2. From this property it follows that all permutations are equally likely to be the least-cost permutation initially. There are $n!$ permutations in S_n and $(n-1)!$ cyclic permutations (we can fix any of the n elements of an n-cycle as a starting point. Thereafter there are $(n-1)!$ ways to arrange the remaining $n-1$ elements to close the cycle). We find then that the probability that the least-cost permutation is cyclic is

$$P_n(0) = \frac{(n-1)!}{n!} = \frac{1}{n}. \tag{6.1}$$

The following theorem helps us obtain asymptotic values for $P_n(k)$ when $k \geq 1$.

THEOREM 6.1. *Let $S(n, k)$ denote the probability that a randomly picked n-permutation is composed of cycles each of order greater than k assuming that all permutations are equally likely. Then*[1]

$$\lim_{n\to\infty} S(n, k) = \exp(-H_k) \qquad \text{for} \quad k \geq 0.$$

PROOF. We proceed by induction on k. First note that by definition the number of n-permutations whose cycles all have order greater than k is $n!S(n, k)$. For the basis of the induction we note that all n-permutations are composed of cycles of order greater than 0. So for all n, $S(n, 0) = 1 = \exp(-H_0)$ and $\lim_{n\to\infty} S(n, 0) = 1 = \exp(-H_0)$.

Assume now that $\lim_{n\to\infty} S(n, k-1) = \exp(-H_{k-1})$ for some $k > 0$. The probability $S(n, k)$ can be formulated as $(1/n!) \times$ (number of permutations whose subcycles all have order greater than k). The essential idea here is to subtract the number of permutations which contain some cycles of order k from the $n!S(n, k-1)$ permutations which have cycles all of order $>k-1$. First of all there are $n!S(n, k-1)$ permutations whose cycles have order greater than or equal to k. Suppose now that we select k nodes (regarding them as material for a cycle of order k). There are $\binom{n}{k}$ ways to select k nodes, $k-1!$ ways to arrange them in a cycle, and there are $(n-k)!S(n-k, k-1)$ ways to form permutations on the remaining $n-k$ nodes such that all cycles have order greater than or equal to k. Suppose next that we select two sets of k nodes. There are $\binom{n}{k}\binom{n-k}{k}$ ways to select them, $(k-1)!(k-1)!/2!$ unique ways to arrange the two sets into two cycles of order k (the divisor 2! is the number of ways of picking the same set of two cycles), and there are $(n-2k)!S(n-2k, k-1)$ permutations of the remaining $n-2k$ nodes such that all cycles have order greater than or equal to k. In general suppose we select m disjoint sets of k nodes and arrange each set into a cycle of order k. There are $\binom{n}{k}\binom{n-k}{k}\cdots\binom{n-mk+k}{k}$ ways to pick m such sets, $(k-1)!^m/m!$ ways to arrange these sets into

[1] The numbers $H_n = \sum_{k=1}^{n} 1/k$ are called harmonic numbers and occur frequently in the analysis of algorithms.

cycles of order k (there is a repetition factor of $m!$ because each particular arrangement of the m cycles can be permuted in $m!$ ways), and finally there are $(n - mk)!S(n - mk, k - 1)$ ways to arrange the remaining $n - mk$ nodes into permutations composed of cycles of order greater than or equal to k.

Applying the principle of inclusion-exclusion [29], we find

$$\begin{aligned}
S(n, k) &= \frac{1}{n!} \sum_{m=0}^{\lfloor n/k \rfloor} (-1)^m \frac{(k-1)!^m}{m!} \binom{n}{k}\binom{n-k}{k} \cdots \binom{n-mk+k}{k} \\
&\qquad \cdot (n - mk)! S(n - mk, k - 1) \\
&= \sum_{m=0}^{\lfloor n/k \rfloor} \frac{(-1)^m}{n!} \frac{(k-1)!^m}{m!} \frac{n!}{k!(n-k)!} \frac{(n-k)!}{k!(n-2k)!} \cdots \frac{(n-mk+k)!}{k!(n-mk)!} \\
&\qquad \cdot (n - mk)! S(n - mk, k - 1) \\
&= \sum_{m=0}^{\lfloor n/k \rfloor} \frac{(-1/k)^m}{m!} S(n - mk, k - 1).
\end{aligned}$$

Taking the limit of this function, we obtain

$$\begin{aligned}
\lim_{n\to\infty} S(n, k) &= \lim_{n\to\infty} \sum_{m=0}^{\lfloor n/k \rfloor} \frac{(-1/k)^m}{m!} S(n - mk, k - 1) \\
&= \sum_{m=0}^{\infty} \frac{(-1/k)^m}{m!} \lim_{n\to\infty} S(n - mk, k - 1) \\
&= \sum_{m=0}^{\infty} \frac{(-1/k)^m}{m!} \exp(-H_{k-1}) \quad \text{(by induction hypothesis)} \\
&= \exp(-H_{k-1}) \exp(-1/k) \\
&= \exp(-H_k).
\end{aligned}$$

Q.E.D.

An immediate corollary of Theorem 6.1 is the well-known result that the number of n-permutations which do not have any 1-cycles is $n!S(n, 1)$ which is asymptotic to $n!\exp(-H_1) = n!/e$ (this is known as the problem of derangements [29]). Our intended application of Theorem 6.1 is the probability that the least-cost permutation has k sons (its smallest subcycle is of order k).

THEOREM 6.2. *The asymptotic probability that the smallest order cycle of a random permutation has order k is*

$$\lim_{n\to\infty} P_n(k) = \exp(-H_{k-1}) - \exp(-H_k). \tag{6.2}$$

PROOF. The probability that a random permutation π has a smallest subcycle of order k is the probability that the cycles of π have order greater than $k - 1$, minus the probability that the subcycles of π have order greater than k. The theorem then follows directly from Theorem 6.1. Q.E.D.

The growth of

$$\bar{P}_n = \sum_{k=1}^{\lfloor n/2 \rfloor} k P_n(k)$$

has been shown in [35] to be asymptotic to $\exp(-\gamma)\ln(n)$, where $\gamma = 0.577 \ldots$ is Euler's constant, which can also be seen by considering the asymptotic growth of

$$\sum_{k=1}^{\lfloor n/2 \rfloor} k[\exp(-H_{k-1}) - \exp(-H_k)]. \tag{6.3}$$

The formulas (6.1) and (6.2) describe the behavior of a subtour-elimination algorithm on a random instance of the TSP, since any permutation is equally likely to be the least-cost permutation. Assuming that it can be shown that for random subinstances the expected branching factor is $O(\ln(n))$, and the probability that the relaxed solution is cyclic grows no slower than $1/n$, then by Theorem 4.1 the expected search tree size for a random TSP instance is $O(n \ln(n))$. The running time of BB on a random instance is dominated by two factors. First, the time spent solving the assignment problem is $O(n^3)$ for the initial instance and $O(n^2)$ for all other subinstances in the tree. The net contribution of these terms is

$$O(n^3) + O(n \ln(n)) \cdot O(n^2) = O(n^3\ln(n)).$$

Second, when there are m objects in the priority queue, $O(\ln(m))$ time is sufficient for both insertion and deletion. It is shown in Appendix B that the mean queue maintenance time for a random TSP is $O(n^2\ln^2(n))$. Thus the expected running time of a subtour-elimination algorithm is

$$O(n^3\ln(n)) + O(n^2\ln^2(n)) = O(n^3\ln(n)). \tag{6.4}$$

(6.4) is consistent with empirically obtained estimates of the expected running time of subtour-elimination algorithms. Bellmore and Malone [4] report $O(n^{3\,46})$ expected running time over the range $10 \le n \le 80$, and $O(n^{3\,2})$ is given in [37] for the range $30 \le n \le 200$. It has been pointed out in [28] that establishing $O(n^{-c})$, for some constant c, as a lower bound on the probability that a random subinstance yields a feasible relaxed solution is sufficient to establish polynomial expected running time for these algorithms. Note that the average branching factor must be $O(n)$. While the evidence suggests such a result, it remains an open theoretical problem.

The probability that the least-cost n-permutation has a 1-cycle for large n is roughly $\exp(-H_0) - \exp(-H_1) = 1 - 1/e \approx 0.63$. Since a traveling salesman tour cannot have any 1-cycles, if we insert infinites along the diagonal of our random cost matrices we do not lose any cyclic permutations, yet we reduce the size of the relaxed space by about 63 percent. Unfortunately, there is no readily apparent analogous method for precluding permutations with 2-cycles (or higher order cycles). We can estimate the probability that a cyclic permutation is optimal with respect to the altered matrix as

$$P'_n(0) = \frac{(n-1)!}{(n!/e)} = \frac{e}{n}. \tag{6.5}$$

It cannot be shown that (6.5) is asymptotically correct as easily as (6.1) can be shown correct because the set of permutations without 1-cycles is not symmetric in the sense given above. Nonetheless, observations of randomly generated traveling salesman problems given in Table I supports (6.5). With respect to the modified matrix, the probability $P'_n(k)$ that the optimal solution to a random instance has a smallest order cycle of order k is estimated by $P_n(k)/(1 - P_n(1))$ which converges to the value

$$\frac{(\exp(-H_{k-1}) - \exp(-H_k))}{((1 -)(1 - 1))/e} = e(\exp(-H_{k-1}) - \exp(-H_k)). \tag{6.6}$$

A simple estimate of $\bar{P}'_n$ is

$$\exp(1 - \gamma)\ln(\lfloor n/2 \rfloor - 1) \tag{6.7}$$

TABLE I. Best-Bound-First Search

Size of problem	Number of problems solved	Sample mean search tree size	Mean search tree size by (6.8)	Sample $\bar{P}$ at root	Estimate of $\bar{P}$ by (6.7)	Sample $P(0)$ at root	$P(0)$ by eq. 6.5 ($= e/n$)
10	1000	6.48	8.78	2.03	2.12	0.261	0.262
15	1000	12.28	16.11	2.58	2.74	0.186	0.181
20	1000	19.63	25.66	3.09	3.35	0.153	0.136
25	790	31.19	34.58	3.50	3.66	0.106	0.109

TABLE II. Depth-First Search[a]

	Sample values of P at depth m in the search tree										
	0	1	2	3	4	5	6	7	8	9	10+
$P(0)$	0.135	0.230	0.298	0.322	0.331	0.349	0.354	0.356	0.364	0.373	0.368
$P(2)$	0.386	0.195	0.141	0.122	0.115	0.101	0.090	0.099	0.084	0.090	0.097
$P(3)$	0.180	0.159	0.133	0.123	0.115	0.106	0.105	0.098	0.099	0.101	0.088
$P(4)$	0.105	0.108	0.109	0.102	0.097	0.096	0.090	0.089	0.087	0.081	0.089
$P(5)$	0.066	0.085	0.086	0.082	0.081	0.084	0.087	0.091	0.098	0.096	0.096
$P(6)$	0.037	0.055	0.065	0.068	0.071	0.072	0.074	0.066	0.067	0.067	0.070
$P(7)$	0.032	0.049	0.049	0.056	0.056	0.060	0.061	0.054	0.065	0.051	0.047
$P(8)$	0.030	0.050	0.050	0.052	0.053	0.055	0.057	0.062	0.055	0.055	0.056
$P(9)$	0.020	0.042	0.046	0.049	0.053	0.052	0.054	0.060	0.055	0.054	0.060
$P(10)$	0.009	0.026	0.023	0.023	0.028	0.026	0.027	0.024	0.025	0.033	0.031
$\bar{P}$	3.018	3.435	3.324	3.338	3.367	3.344	3.373	3.353	3.345	3.302	3.341

[a] Data from the solution of 790 randomly generated asymmetric traveling salesman problems with 20 nodes by a subtour-elimination algorithm using a depth-first search strategy and given an initial bound of 1000 plus the lower bound on the root. Sample values of the probability function P at various depths in the search tree are given. At the bottom of each column is the sample mean of P for nodes found at that depth. The last column summarizes data on nodes of depth 10 or more.

which is the asymptotic value of

$$= \sum_{k=2}^{n/2} ke(\exp(-H_{k-1}) - \exp(-H_k)).$$

Plugging our estimates (6.5) and (6.7) into the expression $1 + \bar{P}/P(0)$ from Theorem 4.1 we obtain

$$E(N_T) \approx 1 + \exp(-\gamma)n \ln(\lfloor n/2 \rfloor - 1). \tag{6.8}$$

In Table I the estimate (6.8) is computed for several values of n. Compared with these values are empirical values of $E(N_T)$ found by averaging N_T over randomly generated traveling salesman instances for each value of n solved by the subtour-elimination algorithm under a best-bound-first search strategy. Random cost matrices were generated by putting independently and uniformly distributed random integers between 1 and 1000 in each entry. The diagonal entries were set to a very large number.

Table II presents data on the probablities of the various branching factors of nodes at different depths in the search tree. Notice that $P_n(0)$ appears to increase monotonically with depth. This provides evidence that e/n is indeed a lower bound on $P_n(0)$. The most dramatic changes take place between depth 0 and depth 1. In particular, $P_n(0)$ almost doubles and $P_n(2)$ roughly halves. Below depth 1 there is relative stability of the sample probabilities and sample mean. The fact that the

TABLE III. DATA FROM DEPTH-FIRST SEARCH USING AN INITIAL BOUND OF 1000 + THE VALUE OF THE LOWER BOUND ON THE ROOT

Size of problem	Number of problems solved	Sample mean search tree size	ET bound = 1000	ET' bound = 1000
10	1000	10.36	11.06	13.45
15	1000	35.82	30.03	38.61
20	790	81.85	64.40	88.72

TABLE IV. DEPTH-FIRST SEARCH USING AN INITIAL BOUND OF ∞

Size of problem	Number of problems solved	Mean search tree size when the leftmost branch has length d										
		0	1	2	3	4	5	6	7	8	9	10
10	1000	1	5	12	20	27	36	43	51	—		
15	1000	1	14	35	48	72	89	99	111	125	145	—
20	790	1	17	35	85	94	128	160	182	207	299	—

estimate (6.8) provides an upper bound on the sample data seems to be due to the increase in $P_n(0)$ with depth.

In order to predict some of the properties of a depth-first search on traveling salesman instances, we need a way of estimating the probability function for the arc cost Q_n. We have found empirically that Q_n is estimated by the geometric function

$$Q_n(k) = (0.000054n)\left(\frac{1}{1 + 0.00054n}\right)^k. \tag{6.9}$$

Table III compares some sample mean time complexity statistics for randomly generated traveling salesman problems solved using a depth-first search with estimates generated by the function ET introduced in Section 5. The randomly generated problems were given an initial bound of 1000 (actually 1000 + lower bound on the initial feasible set) and the recurrence relation for ET was computed out to $ET(1000)$. We used (6.9) for Q_n and our formulas (6.5) and (6.6) for P_n' in computing ET, in the column marked $ET(1000)$. Note that $ET(1000)$ using this P_n' function underestimates the sample mean. We obtain a better estimate by amending P'_n as follows: Halve $P_n'(2)$ and distribute the difference over $P_n'(3)$, $P_n'(4), \ldots, P_n'(\lfloor n/2 \rfloor)$. We retain $P_n'(0) = e/n$. In this way the mean of $\bar{P}_n'$ has been increased and the lower bound on $P_n'(0)$ remains (cf. Theorem 4.1). The bounds obtained using this P_n' function in ET are given in the column labeled $ET'(1000)$ in Table III.

Theorem 5.3 predicts that the expected size of the search tree in a depth-first-search grows essentially linearly as a function of the length of the left-most path in the search tree. Empirical data gathered from random TSP instances are presented in Table IV and Figure 5. The data in Figure 5 clearly show the linear growth of the mean search tree size for as far as the sample means are meaningful.

7. *Concluding Remarks*

A random process for generating arc-labeled trees has been defined and some of its properties have been developed. This process has served as a model of the kind

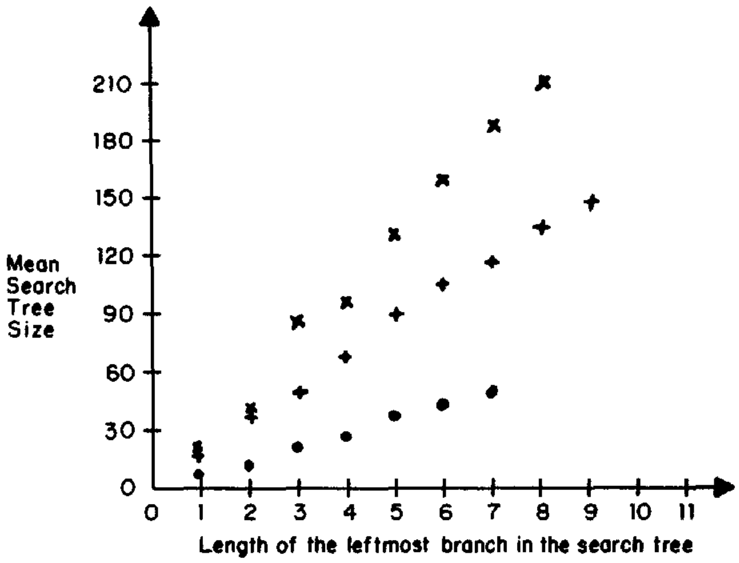


FIG 5. The data from Table IV plotted—showing the growth of the sample mean search tree size as a function of the length of the leftmost branch in the search tree The circles, pluses, and x's represent data points from the traveling salesman problems of size 10, 15, and 20, respectively.

of trees generated by a branch and bound procedure and has enabled the derivation of a number of formulas for the expected space and time complexities of a branch and bound procedure under several search strategies. In particular, it has been shown that the best-bound-first search strategy is optimal in both time and space complexity. These results, together with the simplicity of the basic branch and bound procedure in Section 2, and the existence of efficient techniques for implementing priority queues [1], strongly suggest the use of the best-bound-first search strategy in applications.

A similar random process has been used by Lapin [23] to model branch and bound procedures which seek the least-cost node at a fixed depth. The class of search strategies investigated are realized by the heuristic function

$$h(N) = \mathrm{LB}(N) - \alpha \mathrm{depth}(N)$$

where α is a parameter. Solutions are guaranteed to be optimal iff α is zero. Generating functions are derived for the number of instances to which the branching function is applied and for the distribution of solution costs found by the algorithm. In principle, one could derive an expression for the expected value of the time complexity from the former generating function, but it appears to be difficult to do so, and Lapin does not carry out this exercise.

Variations of our model can provide models for other procedures whose essential nature is tree searching. In many branch and bound procedures a solution is found at some fixed depth k. Several types of models for this situation were mentioned in Section 2. Game trees can be modeled by (P, Q)-trees in which the domain of Q is $[-c, c]$ for some positive constant c. A negative arc cost corresponds to a move in which the resulting board is evaluated as being less good than the starting position. An advantage to this model, in comparison to other models of game trees which have been explored [21], is that dependencies between moves are propagated in the tree, that is, all moves following a bad move (corresponding to a negative arc cost) would tend to have lower evaluations than the moves following a good move (corresponding to a positive arc cost). Another advantage is that randomness in the number of legal (or plausible) moves is part of the model.

Our model is particularly suited for modeling relaxation-guided procedures, where there is some chance that any node in the search tree of a random instance of a problem may produce a feasible solution. The success of the assignment problem relaxation for solving asymmetric traveling salesman problems and Held and Karp's 1-tree relaxation for solving symmetric traveling salesman problems suggests that the search for polynomial expected time algorithms for solving NP-hard problems might begin by looking for suitable relaxations and fast algorithms for solving them. The search for fast approximate algorithms for NP-hard problems can also benefit from the use of relaxations of a problem. A relaxed solution to an instance may have many of the components of an optimal feasible solution, thus a heuristic restructuring of the relaxed solution might produce a feasible solution of near optimal cost [19].

Appendix A

Several of the results of this paper have been formulated as somewhat complex recurrence relations. We show how two of these recurrence relations can be broken down into simpler relations which aid in the computation of their sequences. In Section 3 the function $\hat{O}$ was introduced in the form

$$1 - \sum_{k=0}^{i} \hat{O}(k) = \sum_{j=1}^{\infty} P(j)\left[1 - \sum_{s=1}^{i}\sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}$$

with boundary condition $\hat{O}(0) = P(0)$.

Let

$$\begin{aligned} E(s) &= \sum_{c=1}^{s} Q(c)\hat{O}(s-c), \\ G(i) &= 1 - \sum_{s=1}^{i}\sum_{c=1}^{s} Q(c)\hat{O}(s-c), \\ &= 1 - \sum_{s=1}^{i} E(s) = G(i-1) - E(i), \\ B(i) &= \sum_{j=1}^{\infty} P(j)G(i)^{j}, \\ \hat{O}(i) &= B(i-1) - B(i). \end{aligned}$$

Note that $B(i) = 1 - \sum_{k=0}^{i} \hat{O}(k)$; therefore $B(i-1) - B(i) = \hat{O}(i)$. The sequence $\{\hat{O}(i)\}_{i=0,n}$ may be computed as follows (assuming a suitable bounding of the computation of $B(i)$).

```
begin
$G(0) := 1$;
$B(0) := 1 - P(0)$;
$\hat{O}(0) := P(0)$;
for $i := 1$ until $n$;
  begin
    $E(i) := \sum_{c=1}^{i} Q(c)\hat{O}(i-c)$;
    $G(i) := G(i-1) - E(i)$;
    $B(i) := \sum_{j=1}^{\infty} P(j)G(i)^{j}$;
    $\hat{O}(i) := B(i-1) - B(i)$;
  end
end
```

The recurrence relation for $ET(b)$ introduced in Section 6 can be simplified in a similar manner. $ET(b)$ has the form

$$ET(b) = 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} W(b, i) \tag{A1}$$

where

$$W(b, i) = \sum_{c_1=1}^{\infty} \cdots \sum_{c_i=1}^{\infty} \sum_{m_1=0}^{\infty} \cdots \sum_{m_{i-1}=0}^{\infty} Q(c_1) \cdots Q(c_i)\hat{O}(m_1) \cdots \hat{O}(m_{i-1})$$
$$\cdot\ ET(\min\{b, c_1 + m_1, c_2 + m_2, \ldots, c_{i-1} + m_{i-1}\} - c_i).$$

Essentially, $W(b, i)$ has the form

$$W(b, i) = \sum_{k=1}^{\infty} R(b, i, k) \sum_{c_i=1}^{\infty} Q(c_i)ET(k - c_i) \tag{A2}$$

where $R(b, i, k)$ = probability that $k = \min\{b, c_1 + m_1, \ldots, c_{i-1} + m_{i-1}\}$. (The term $c_j + m_j$ is the cost of the least-cost leaf in the jth subtree below the root; c.f. Figure 2a.) In other words, k is the value of the bound immediately after the i − 1st subtree has been explored. $R(b, i, k)$ may be formulated easily as follows: We have two cases, either $k = b$ or $k < b$. The probability that $k = b$ is

$$R(b, i, b) = \Pr(c_1 + m_1 \geq b)\Pr(c_2 + m_2 \geq b)\cdots\Pr(c_{i-1} + m_{i-1} \geq b).$$

Again let

$$E(s) = \sum_{c=1}^{s} Q(c)\hat{O}(s - c)$$
$$G(k) = 1 - \sum_{s=1}^{k-1} \sum_{c=1}^{s} Q(c)\hat{O}(s - c)$$
$$= 1 - \sum_{s=1}^{k-1} E(s) = G(k - 1) - E(k - 1).$$

Here $G(k) = \Pr(c + 1 \geq k)$ and $E(k) = \Pr(c + 1 = k)$, so $R(b, i, b) = G(b)^{i-1}$. (A3)

The other case we need to consider occurs when one of the subtrees contains a least-cost leaf which improves the initial bound b. The probability that the bound has the value m is the probability that one of the subtrees has a least-cost leaf of cost m and the rest have least-cost leaves of cost $\geq m$; thus, noticing that each of the $i - 1$ subtrees may contain the least-cost leaf, we have,

$$R(b, i, m) = (i - 1)E(m)G(m)^{i-2}. \tag{A4}$$

Substituting (A3) and (A4) into (A2) we get

$$W(b, i) = \sum_{k=1}^{b-1} (i - 1)E(k)G(k)^{i-2}D(k) + G(b)^{i-1}D(b)$$

where

$$D(k) = \sum_{c=1}^{k} Q(c)ET(k - c).$$

Further, letting

$$H(b, i) = \sum_{k=1}^{b-1} (i - 1)E(k)G(k)^{i-2}D(k)$$
$$= H(b - 1, i) + (i - 1)E(b - 1)G(b - 1)^{i-2}D(b - 1),$$

we have

$$W(b, i) = H(b, i) + G(b)^{i-1}D(b).$$

Looking again at (A1), we see that we need partial sums of $W(b, i)$, so let

$$\begin{aligned} V(b, i) = \sum_{j=1}^{i} W(b, i) &= V(b, i-1) + W(b, i) \\ &= V(b, i-1) + H(b, i) + G(b)_{i-1}D(b). \end{aligned}$$

Putting all these pieces together, we can compute $\{ET(b)\}_{b=0,n}$ as follows.

```
begin
$ET(0) := 1$;
for all $b$, $V(b, 0) := 0$;
for all $b$, $H(b, 0) := 0$;
$G(0) := 1$;
$E(0) := 0$;
for $b := 1$ until $n$ do
  begin
    for $i := 1, \ldots, \infty$
      $H(b, i) := H(b, i-1) + (i-1)E(b-1)G(b-1)^{i-2}D(b-1)$;
    $G(b) := G(b-1) - E(b-1)$;
    $E(b) := \sum_{c=1}^{b} Q(c)\hat{O}(b-c)$;
    $D(b) := \sum_{c=1}^{b} Q(c)ET(b-c)$;
    for $i := 1$ until $\infty$;
      $V(b, i) := V(b, i-1) + H(b, i) + G(b)^{i-1}D(b)$;
    $ET(b) := 1 + \sum_{j=1}^{\infty} P(j)V(b, j)$;
  end
end
```

The infinities which appear in the algorithms of Figures A1 and A2 only come into play when P has an infinite range, that is, arbitrarily large branching factors are possible. In most practical classes of problems the branching factor is in fact bounded. When modeling such cases the infinities are replaced by whatever bound exists on the branching factor. In an implementation of this algorithm, the arrays E, G, and D can be replaced by single variables, since only the most recently computed value of the corresponding array is ever used. Similarly the two-dimensional arrays V and H can be reduced to one-dimensional arrays.

Appendix B

In this section we derive a bound on the expected queue maintenance time for a best-bound-first search. The generating function for the random sum (4.1) is $1 + g_n(p_n(z))$ where g_n is the generating function for the number of terms k in (4.1),

$$g_n(z) = \sum_{i=0}^{\infty} (1 - P_n(0))^i P_n(0) z^i$$

and $p_n(z)$ is the generating function for P_n,

$$p_n(z) = \sum_{i=0}^{\infty} P_n(i) z^i.$$

Letting $R_n(i)$ denote the probability that $N_T = i + 1$, we have

$$g_n(p_n(z)) = \sum_{i=0}^{\infty} R_n(i) z^i.$$

When the search tree size is $N_T = i + 1$, we have at most $2(i + 1)$ insertions and deletions from the priority queue which takes $O(2(i + 1)\ln(2(i + 1))) = O(i \ln(i))$ time [1]. To within a constant factor the expected queue maintenance time then is

$$\sum_{i=1}^{\infty} i \ln(i) R_n(i). \tag{B1}$$

A bound on (B1) can be obtained from the generating function

$$h_n(z) = z^2 \frac{d^2}{dz} g_n(p_n) = \sum_{i=0}^{\infty} i(i - 1)R_n(i)z^i.$$

Note that (B1) is bounded by $h_n(1)$. Performing the differentiation we find

$$h_n(1) = g_n'(1)p_n'(1)p_n'(1) + p_n''(1)g_n'(1)$$

where $f'(z)$ denotes $(d/dz)f(z)$ and $f''(z)$ denotes $(d^2/dz)f(z)$. Using the estimates $P_n(0) = e/n$, and $P_n(k) = (\exp(-H_{k-1}) - \exp(-H_k))$, we find

$$\begin{aligned}
g_n'(1) &= \frac{1 - P(0)}{P(0)} = \frac{n}{e}\left(1 - \frac{e}{n}\right) = O(n),\\
g_n''(1) &= 2g_n'(1)^2 = O(n^2),\\
p_n'(1) &= \bar{P}_n = O(\ln(n)),\\
p_n''(1) &= \sum_{k=1}^{n/2} k(k - 1)P_n(k) = \sum_{k=1}^{n/2} k(k - 1)e(\exp(-H_{k-1}) - \exp(-H_k))\\
&= 2e \sum_{k=1}^{n/2} k \exp(-H_k) \le 2e \sum_{k=1}^{n/2} k \exp(-\gamma - \ln(k))\\
&= 2 \exp(1 - \gamma)\left(\frac{n}{2}\right) = O(n).
\end{aligned}$$

Thus

$$h_n(1) = O(n^2\ln^2(n)) + O(n^2) = O(n^2\ln^2(n)).$$

ACKNOWLEDGMENTS. Most of the results presented above originally appeared in the author's dissertation [36], supervised by Alan Biermann. His interest in search theory helped motivate this research. The referees provided several suggestions which led to improvements in the correctness and readability of the manuscript.

REFERENCES

1. AHO, A V., HOPCROFT, J.E., AND ULLMAN, J.D. *The Design and Analysis of Algorithms*. Addison-Wesley, Reading, Mass., 1974.
2. BALAS, E. A note on the branch and bound principle. *Oper Res 16* (1968), 442–445.
3. BELLMORE, M., AND NEMHAUSER, G.L. The traveling salesman problem: A review. *Oper. Res 16* (1968), 538–558.
4. BELLMORE, M., AND MALONE, J.C. Pathology of traveling salesman subtour-elimination algorithms. *Oper Res. 19* (1971), 278–307.
5. FOX, B.L., LENSTRA, J.K., RINNOOY KAN, A.H.G., AND SCHRAGE, L.E. Branching from the largest upper bound. *European J Oper Res 2* (1978), 191–194.
6. GAREY, M.R., AND JOHNSON, D.S. *Computers and Intractability· A Guide to the Theory of NP-Completeness* Freeman, San Francisco, 1979.
7. GARFINKEL, R.S. On partitioning the feasible set in a branch and bound algorithm for the asymmetric traveling salesman problem. *Oper. Res. 21* (1973), 340–343.

8. GARFINKEL, R.S., AND NEMHAUSER, G.L. *Integer Programming* Wiley, New York, 1972.
9. GREENBERG, H., AND HEGERICH, R. A branch search algorithm for the knapsack problem. *Manage. Sci. 16,* 5 (1970), 327–332.
10. HARRIS, T.E. *The Theory of Branching Processes.* Springer-Verlag, Berlin, 1963.
11 HELD, M., AND KARP, R.M. The traveling salesman problem and minimum spanning trees. *Oper. Res. 18* (1970), 1138–1162.
12. HELD, M., AND KARP, R.M. The traveling salesman problem and minimum spanning trees: Part II. *Math. Program. 1* (1971), 6–25.
13. HOROWITZ, E., AND SAHNI, S. Computing partitions with applications to the knapsack problem. *J ACM 21,* 2 (Apr. 1974), 277–292.
14. IBARAKI, T Theoretical comparison of search strategies in branch and bound algorithms. *Int J Comput. Inf. Sci. 5,* 4 (1976), 315–344
15. IBARAKI, T. The power of dominance relations in branch-and-bound algorithms. *J ACM 24,* 2 (Apr. 1977), 264–269.
16. IBARAKI, T. Branch and bound procedure and state-space representation of combinatorial optimization problems. *Inf. Control 36* (1978), 1–27.
17 IGNALL, E., AND SCHRAGE, L. Application of the branch and bound technique to some flow-shop scheduling problems. *Oper Res 11* (1965), 400–412.
18. KANAL, L.N. Problem solving models and search strategies for pattern recognition. *IEEE Trans. Pattern Anal Mach. Intell 1,* 2 (Apr. 1979), 193–201.
19. KARP, R.E. A patching algorithm for the nonsymmetric traveling salesman problem. *SIAM J Comput 8,* 4 (Nov. 1979), 561–573.
20. KLEE, V., AND MINTY, G.J. How good is the simplex algorithm? Math. Note No. 643, Boeing Scientific Research Laboratories, 1970.
21. KNUTH, D.E., AND MOORE, R.W. An analysis of alpha-beta pruning. *Artif Intell. 6* (1975), 293–326.
22. KOHLER, W.H., AND STEIGLITZ, K. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *J. ACM 21,* 1 (Jan. 1974), 140–156.
23. LAPIN, YU. P. Probability modeling of branch and bound method. *Cybernetics 3,* 16 (May-June 1980), 428–434.
24. LAWLER, E.L. The quadratic assignment problem. *Manage Sci. 9,* 4 (July 1963), 586–599.
25. LAWLER, E.L., AND WOOD, D.E. Branch and bound methods: A survey. *Oper Res 14,* 4 (1966), 699–719.
26. LAWLER, E.L. *Combinatorial Optimization Networks and Matroids* Holt, Rinehart, and Winston, New York, 1976.
27. LEMKE, C.E., SALKIN, H.M., AND SPIELBERG, K. Set covering by single branch enumeration with linear programming subproblems. *Oper Res 19* (1971), 998–1022.
28. LENSTRA, J.K., AND RINNOOY KAN, A.H.G. On the expected performance of branch and bound algorithms. *Oper. Res 26,* 2 (1978), 347–349.
29. LIU, C.L. *Introduction to Combinatorial Mathematics.* McGraw-Hill, New York, 1968.
30. MARTELLI, A., AND MONTANARI, U. Optimizing decision trees through heuristically guided search. *Commun ACM 21* (1978), 1025–1039.
31. MITTEN, L G. Branch and bound methods: general formulation and properties. *Oper Res. 18* (1970), 24–34.
32. NILSSON, N.J. *Problem Solving Methods in Artificial Intelligence* McGraw-Hill, New York 1971.
33. RINNOOY KAN, A.H.G. On Mittens' axioms for branch and bound. Working Paper W/74/45/03, Graduate School of Management, Delft, The Netherlands, 1974.
34. SHAPIRO, D.M. Algorithms for the solution of the optimal cost and bottleneck traveling salesman problems. Unpublished Sc.D. dissertation, Washington Univ., St. Louis, Mo., 1966.
35. SHEPP, L.A., AND LLOYD, S.P. Ordered cycle length in a random permutation. *Trans. Am. Math Soc 121,* 2 (Feb. 1966), 340–357.
36 SMITH, D.R. On the computational complexity of branch and bound search strategies. Ph.D. dissertation, Duke Univ., 1979. Available as Tech. Rep. NPS 52-79-004, Dept. of Computer Science, Naval Postgraduate School, Monterey, Calif.
37 SMITH, T.H.C., SRINIVASAN, V., AND THOMPSON, G.L. Computational performance of three subtour-elimination algorithms for solving traveling salesman problems. *Ann Discrete Math 1* (1977), 495–506.

RECEIVED MARCH 1981; REVISED NOVEMBER 1982; ACCEPTED DECEMBER 1982