1979-11

On the computational complexity of branch and bound search strategies

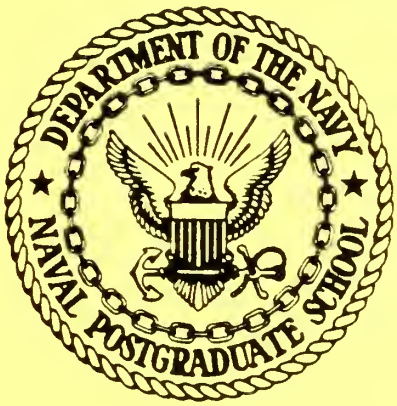
Smith, Douglas R.

Monterey, California. Naval Postgraduate School

NPS 52-79-004

NAVAL POSTGRADUATE SCHOOL

Monterey, California



On the Computational Complexity
of Branch and Bound Search Strategies

by

Douglas R. Smith

November 1979

Approved for public release; distribution unlimited.

epared for:
tional Science Foundation
shington, D. C. 20550

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral T. F. Dedman
Superintendent

Jack R. Borsting
Provost

This research was partially supported by the National Science Foundation.

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS 52-79-004
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): On The Computational Complexity Of Branch And Bound Search Strategies
5. TYPE OF REPORT & PERIOD COVERED: Technical Report
6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Douglas R. Smith
8. CONTRACT OR GRANT NUMBER(s): NSF Grant MCS74-14445-A01
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93940
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS: National Science Foundation, Washington, D. C. 20550
12. REPORT DATE: November 1979
13. NUMBER OF PAGES: 100
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report): Approved for Public Release; distribution unlimited
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
Combinatorial Optimization
Branch and Bound
Complexity of Computation
Search Strategy
Tree Search
Probabilistic Modelling
20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

Many important problems in operations research, artificial intelligence, and other areas of computer science seem to require search in order to find an optimal solution. A branch and bound procedure, which imposes a tree structure on the search, is often the most efficient known means for solving these problems. While for some branch and bound algorithms a worst case complexity bound is known, the average case complexity is usually unknown despite the fact that it gives more information about the

performance of the algorithm. In this dissertation the branch and bound method is discussed and a probabilistic model of its domain is given, namely a class of trees with an associated probability measure. The best-bound-first search strategy and depth-first search strategy are discussed and results on the expected time and space complexity of these strategies are presented and discussed. The best-bound-first search strategy is shown to be optimal in both time and space. These results are illustrated by data from randomly generated traveling salesman problems. Evidence is presented which suggests that the assymetric traveling salesman problem can be solved in time $O(n^3 \ln^2(n))$ on the average.

ABSTRACT

Many important problems in operations research, artificial intelligence, combinatorial algorithms, and other areas seem to require search in order to find an optimal solution. A branch and bound procedure, which imposes a tree structure on the search, is often the most efficient known means for solving these problems. While for some branch and bound algorithms a worst case complexity bound is known, the average case complexity is usually unknown despite the fact that it gives more information about the performance of the algorithm. In this dissertation the branch and bound method is discussed and a probabilistic model of its domain is given, namely a class of trees with an associated probability measure. The best bound first and depth-first search strategies are discussed and results on the expected time and space complexity of these strategies are presented and compared. 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 problems. Evidence is presented which suggests that the assymetric traveling salesman problem can be solved exactly in time $O(n^3 \ln^2(n))$ on the average.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to Prof. Alan Biermann for his support, constant encouragement, and advice which always seemed to be on target. I would also like to thank Prof. Kishor Trivedi who suggested several key ideas in chapter 4. I am grateful to my wife, Carol, who helped me out in many ways during the course of this work. This dissertation was partially supported by NSF Grant MCS74-14445-A01.

Chapter 1.

Introduction

By an instance of a combinatorial problem we mean the problem of finding a constructive proof of

$$\exists x \; P(x) \quad \text{for } x \in FS \tag{1}$$

where FS is a discrete set of objects and P is a predicate defined on FS. That is, we want to find an object in FS which has the property P. In many cases it is not the truth but rather the feasibility of a constructive proof of (1) which is in doubt. Some combinatorial problems require all solutions which satisfy (1). We restrict ourselves to the problem of finding a single solution, but note that all results obtained in this case can be extended to handle this slighty harder problem. There are countless examples which satisfy (1) ranging from easy problems like sorting (find a permutation of an input list which is sorted), to more difficult problems like integer programming (find an vector of integers which satisfies a set of constraints) and theorem proving (find a proof sequence for a statement in some language by means of a given set of axioms and rules of inference).

Generally when we speak of a combinatorial problem, we mean a set C of related instances of the form (1). These in-

stances can be classified according to their size, enabling us to speak of a problem instance of size n. The question of what the size of an instance is and how to encode problem instances can be tricky. See [Aho, Hopcroft and Ullman 1974] for a disscusion of encodings in the context of the class of problems called P and NP. For our purposes we will say that a measure of the size of a problem instance has the property that all problem instances with the same size have the same feasible set FS. For example in an instance of the sorting problem, we are given a vector of n numbers. Here n is taken as the size of the instance and the feasible set is the set of permutations of n objects. The predicate P(x) tests whether a permutation x applied to the given vector results in a sorted vector. In an instance of an integer programming problem, we are given a set of constraints on n variables. n is taken as the instance size and the feasible set is the set of all integer vectors of length n. In theorem proving we are given a statement and take its length as the size of the instance. Here the feasible set is the set of all legal proof sequences in the theory. If P contains an optimization clause then (1) is called a combinatorial optimization problem. In this dissertation we will be particularly interested in combinatorial minimization problems in which we seek a constructive proof of

$$\exists x[P(x) \;\&\; \forall y[P(y) \Rightarrow f(x) \leq f(y)]] \quad \text{for } x,y \in FS \qquad (2)$$

where f, called the objective function, maps FS into the nonnegative reals.

Some combinatorial problems can be solved directly; i.e. the solution is reached by a straightforward construction with no backtracking. When no direct constructive method is known, there are three principal search methods for finding solutions to combinatorial problems called enumerative search, local search and global search. In an enumerative search the objects of FS are produced one at a time and tested. The search terminates the first time that P is satisfied. If we are seeking an optimal object then FS must be exhaustively searched and FS must be finite in order to assure termination. Some problems such as that of finding a key in an unordered list require enumerative search. Local search [Reiter and Sherman 1965; Lin 1965; Weiner, Savage, and Bagchi 1973; Papadimitriou and Steiglitz 1977] is usually applied to combinatorial optimization problems and is characterized by a topology or neighborhood structure imposed on the set of objects FS. For any object in the set we can readily find all of its neighbors. A search proceeds by selecting some initial object, picking a neighbor which satisfies P and betters the value of the objective function, then picking a neighbor of the neighbor and so on, until an object is found which is optimal with respect to all of its neighbors. This object is called a local optimum. If the neighborhood structure is exact (a local optimum is a global optimum) then the search can terminate on the first local optimum found. For many problems it is difficult to find or infeasible to use an exact neighborhood structure and so a neighborhood structure with many local optima is used. For ex-

ample it has been shown [Weiner, Savage and Bagchi 1973] that in a exact neighborhood structure for the traveling salesman problem on n cities each object (a cyclic permutation of the n cities) must have at least (n-2)!/2 neighbors therefore rendering exact local search infeasible. If many local optima exist then the best we can do is to restart the search at a new initial object, eventually obtaining a set of local optima from which the best may be picked. These local search methods are analogous to the descent and gradient methods of mathematical programming [Luenberger 1973]. On complex spaces this method is best suited for finding approximate solutions, i.e. objects which are nearly optimal but not neccesarily optimal.

A global search is characterized by the handling of sets of objects rather than single objects at a time as in local search. A powerful form of global search may be described as follows. The problem again is to find an object in a set FS which satisfies P. If such an object cannot be found easily then we generate a set of subproblems by splitting FS into subsets. The i^{th} subproblem has the form,

$$\exists x \; P(x) \quad x \in FS_i \subset FS \tag{3}$$

where $\bigcup_i FS_i = FS$. This process of creating subproblems by means of splitting the feasible set is repeated until a solution is found in one of the subsets (which may not occur until the sets are reduced to singleton sets). A global search is the essence of the well-known backtrack technique [Lehmer 1958; Golumb and Baumert 1965, Knuth 1974] of which branch and bound is a special case. Another kind of global search which is re-

lated to branch and bound is the well-known technique of dynamic programming [Bellman 1957; Morin and Marsten 1976, 1978; Ibaraki 1978].

The question of whether search is the most efficient method for solving some combinatorial problems is a deep one which might be specialized in the well-known P=NP question [Cook 1971;Karp 1972]. Problems which can be solved directly tend to have fast algorithms which run in polynomial time in the problem size. On the other hand search algorithms for a problem tend to have a worst-case running time which includes as a factor the size of FS, the feasible set. The NP-complete problems are a class of problems for which either all or none are solvable by algorithms which run in time given by a polynomial of the problem size. Since the size of the search space FS of the NP-complete problems is superpolynomial (usually either exponential or factorial), and all known deterministic algorithms for NP-complete problems have superpolynomial worst-case time bounds, one might conjecture that the P=NP question is equivalent to the question of whether NP-complete problems require search for their solution. At present global search algorithms of the branch and bound variety are the most efficient known methods for solving NP-hard problems. It may be that in answering the P=NP? question wholly new solution methods will be found which obviate the need for search. However the complexity of some global search algorithms is our best current estimate of the intrinsic complexity of a wide range of important combinatorial problems.

The complexity of a global search 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 recently [Klee and Minty 1970] some examples have been found 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 global search algorithms which can have widely varying behaviors over the set of instances of a problem that the average case complexity gives more information than a worst-case measure about the performance of the algorithm.

Branch and bound is a global search technique applicable to combinatorial minimization problems. In the past decade branch and bound seems to have emerged as the principal method for solving problems of this type which have no direct solution. Just a few of the applications of the branch and bound method include integer programming [Garfinkel and Nemhauser 1972], flow shop and job shop sequencing [Ignall and Schrage 1965], traveling salesman problems [Bellmore and Nemhauser 1968; Bellmore and Malone 1972], heuristic search in the form of the A^* algorithm [Hart, Nilsson, and Raphael 1968; Nilsson 1972], and pattern recognition [Kanal 1978]. The alpha-beta technique used in game playing is an extension of branch and bound to the game tree environment [Knuth and Moore 1975].

Branch and bound algorithms can be roughly classified into two kinds according to properties of the trees they generate. In the first kind, solutions to the problem only occur at or below some fixed depth in the tree depending on the problem size. In this approach a solution is built up a component at a time until a complete object is created. In the second kind of algorithm a solution may be found at any depth of the tree (including the possibility that the solution is found at the root). Relaxation procedures fall in this category. In a relaxation procedure a relaxed version of the problem is solved at each node of the tree. In a relaxation of a combinatorial problem of the form (1) we want a constructive proof of

$$\exists x\ P(x) \quad \text{for } x \in FS' \tag{4}$$

where $FS \subset FS'$. This approach may be useful if there is a fast algorithm for solving (4). If the relaxed solution is also a solution to the restricted problem (the solution is in FS), then we're done, otherwise the relaxed solution is used to create subproblems by splitting FS' into subsets in such a way that the relaxed solution is precluded from further consideration. The i^{th} subproblem has the form

$$\exists x\ P(x) \quad x \in FS'_i \subset FS'. \tag{5}$$

where $FS \subseteq \bigcup_i FS'_i \subset FS'$ (c.f. (3)). It is this second kind of branch and bound algorithm which will be modeled and studied in this dissertation. This is not to say that the results of the dissertation do not apply to algorithms of the first kind but merely that our intent was to study the second kind.

The purpose of this dissertation is to analyze the branch

and bound procedure under several search strategies in order to obtain quantitative estimates of their expected time and space requirements. The results of this analysis provide a framework for analyzing and predicting the expected resource requirements of specific branch and bound algorithms as illustrated in chapter 7. These results may also be used to compare the relative efficiency of search strategies.

In chapter 2 the branch and bound algorithm and its domain are presented and several important properties are derived. In Chapter 3 our model of branch and bound search trees is introduced and properties of the model trees are derived. Also the sense in which we will use the term complexity is developed and discussed. Chapter 4 develops results on the complexity of general search strategies. Chapters 5 and 6 apply these results to the best-bound-first search strategy and the depth-first search strategy respectively. Also in chapter 6 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. Using the results obtained in previous chapters, a subtour elimination algorithm for the traveling salesman problem is modeled in chapter 7 and it is suggested that it has an expected running time of $O(n^3 \ln^2(n))$. The reader may wish to reader chapter 7 in parallel with chapters 5 and 6 in order to see an application of the theorems being developed.

Chapter 2.

Branch and Bound Algorithms

Branch and bound algorithms are designed to solve combinatorial minimization problems. We will denote the set of instances of size n of a combinatorial minimization problem by $C_n = (FS_n, COST_n)$ where FS_n is a countable set of objects called the feasible set and $COST_n$ is a set of cost functions such that any $c \in COST_n$ maps FS_n into nonnegative integers. The parameter n of a class C_n varies over positive integers and is intended as a natural measure of the size of the instances of the problem. We will assume that the cost functions must satisfy the condition that no more than a finite number of objects in FS_n may have a given cost. A problem instance from C_n has the following form: Given $c \in COST_n$, find $s^* \in FS_n$ such that for all $s \in FS_n$ $c(s^*) \leq c(s)$, i.e. find a least cost object in the feasible set. In the following discussion we will omit the subscript on FS and COST when no confusion can arise. The idea of a branch and bound search is to split FS into subsets and to compute a lower bound on the cost of the objects within each subset. Those subsets whose bound exceeds the cost of some known (perhaps nonoptimal) solution can be discarded since they cannot contain an optimal solution. The remaining subsets are repeatedly split and bounded until an object is found whose cost

does not exceed the bound on any subset, hence that object is a minimal cost solution. The special power of the branch and bound method comes from this ability to prune away whole sets of objects when they can be shown not to contain an optimal object. The choice of which of the currently unexamined subsets to examine next is specified by a search strategy.

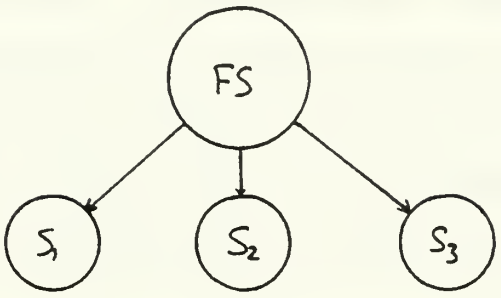
We will use the example of the Traveling Salesman Problem (TSP) throughout this dissertation. The TSP originated with the problem of finding the shortest route for visiting all of n cities and returning to the starting point. We will deal with the following generalization of the TSP of size n. Given a complete directed graph with n nodes and arc weights given by an nxn cost matrix, find the least cost hamiltonian cycle (a cycle which passes once through each node of the graph). Clearly the set of hamiltonian cycles on a complete directed graph is isomorphic to the set of cyclic permutations of n objects. Here the feasible set is the set of all hamiltonian cycles on a complete directed graph of n nodes (or cyclic permutations). The cost functions are a set of cost matrices which specify the arc weights. The cost of a hamiltonian cycle is the sum of the weights on the arcs of the cycle. If $C=[c_{i,j}]$ is a cost matrix, then $c_{i,j}$ is the weight on the directed arc from node i to j. We do not require that $c_{i,j} = c_{j,i}$. There is a long history of attempts to devise efficient algorithms for solving traveling salesman problems. At present the most efficient algorithms for solving TSPs make use of a relaxation procedure embedded in a branch and bound algorithm. The Held

and Karp [Held and Karp 1971] algorithm for solving symmetric TSPs (the cost matrix is required to be symmetric) makes use of a relaxation based on minimum spanning trees. The most efficient known approach to solving assymetric TSPs makes use of a relaxation which allows all permutations to be feasible rather than just cyclic permutations. This relaxed problem is known as the Assignment Problem.

A branch and bound algorithm has three major components. A branching rule B is a rule determining if and how a subset of FS is to be split into subsets. If the least cost object in a subset can be extracted easily then the branching rule does not split the subset. Otherwise the subset is split into a finite number of proper subsets which then represent smaller and therefore easier subproblems to solve. Note that the repeated application of the branching rule generates a tree structure as in figure 2.1. The branching rule for a branch and bound algorithm employing a relaxation procedure is slightly different from branching rules for ordinary branch and bound algorithms. Given a set S, in the latter case $\cup B(S) = S$, and in the former case $\cup B(S) \subset S$ since we preclude some of the relaxed feasible objects. The branching rule of course does not split a singleton set since the least cost object (namely the only object) in the set can be easily extracted. It will be useful to define the function parent as follows: if $S_1 \in B(S)$ for some $S \subseteq FS$ then $parent(S_1)=S$.

The second component of a branch and bound algorithm is a

Figure 2.1. Application of the branching rule to FS.



$B(FS) = \{S_1, S_2, S_3\}$

where $S_1 \subset FS$, $S_2 \subset FS$, $S_3 \subset FS$,

and $S_1 \cup S_2 \cup S_3 \subseteq FS$.

lower bound function which maps subsets of FS into nonnegative integers. Intuitively the lower bound function computes a lower bound on the cost of all objects in a given subset of FS. Formally LB must satisfy the following conditions:

1. for $S \subseteq FS$ and $s \in S$ $LB(S) \leq c(s)$
   (LB computes a lower bound on the cost of objects in S),
2. for $S_i \subseteq S_j \subseteq FS$ $LB(S_j) \geq LB(S_i)$
   (the lower bound values increase monotonically on any path from the root in the tree),
3. if $B(S) = S$ (B does not split S) then $LB(S) = c(s^*)$, where s^* is the least cost object in S
   (when the least cost object in a set can be extracted, the cost of that object is the lower bound value of the set).

The lower bound function is used to eliminate from consideration those subsets of FS which can be shown not to contain the least cost solution. If it is known that a least cost object has a cost of at most c_1 then any subset S for which $LB(S) > c_1$ cannot yield the optimal solution.

The third component of a branch and bound algorithm is a search strategy which is a rule for choosing to which of the currently active subsets of FS the branching rule should be applied . For conceptual simplicity and uniformity of notation, a search strategy will be realized here by a heuristic function $h:2^{FS}$->PRIORITY where PRIORITY is a set which depends on the particular search strategy. Of those subsets waiting to be explored via the branching rule we choose that subset S with the smallest heuristic value h(S). At any particular time during a branch and bound search, a certain set of subsets are waiting to have the branching rule applied to them. If the heuristic value of these subsets (computed by h) are distinct for all such times during a search of any problem in any class then the heuristic function is called unambiguous. Some common search strategies will be discussed below along with unambiguous heuristic functions which realize them.

The branch and bound algorithm for finding a single least cost object is given below in an ad-hoc ALGOL-like language. The principal data structure employed is a priority queue. A priority queue used here is a data structure which stores data objects (in this case nodes representing subsets of FS) with an associated priority given by the heuristic function h. The queue is accessible only by the functions NONEMPTY, which returns true if and only if the queue is nonempty, REMOVETOP, which removes and returns the data object in the queue of highest priority (priority i is higher than priority j if and only if $i \leq_h j$ for a suitable definition of the relation $\leq_h$), and

INSERT which inserts a data object into the queue with its associated priority. Efficient algorithms for manipulating priority queues in this manner are discussed in [Aho, Hopcroft, and Ullman 1974]. The procedure BB in figure 2.2 is typically invoked with the node representing FS and ∞ as arguments. In later discussion of BB we will use the node symbol $N_\emptyset$ to represent FS. An obvious improvement of BB is to check that $cost(N_i) < bound$ in statement 10 before the node N_i is inserted in the 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. Several other enhancements of the pruning power of BB may be added to this code but they are not always easy to discover for a particular problem. A dominance relation [Kohler and Steiglitz 1974; Ibaraki 1977, 1978] is a relation on subsets of FS such that if S_1 dominates S_2 then S_2 cannot contain a better solution than S_1, so S_2 can be eliminated. This test is a direct generalization of the lower bound test. If it can be determined that two subsets $S_1, S_2 \subseteq FS$ are equivalent in the sense that the optimal solution in one is as good as the optimal solution in the other, then only one of these subsets needs to be explored. This test is called an equivalence test [Ibaraki 1977, 1978].

One focus of this dissertation is on several common search strategies and the effect they have on the average case

Figure 2.2. A branch and bound procedure.

```
 1. node procedure BB(node N, integer bound);
      begin priority queue PQ;
        integer i,k;
        node solution;
        integer function REMOVETOP(priority queue),
          INSERT(priority queue,node,PRIORITY), cost(node);
        PRIORITY function h(node);
        boolean function NONEMPTY(priority queue);

 2.     INSERT(PQ,N,h(N));  /* insert root on queue */
 3.     while( NONEMPTY(PQ) ) do
 4.        begin N:=REMOVETOP(PQ);
 5.          if cost(N) < bound
                then begin
 6.             apply the branching rule to N, i.e.
                determine the sons N_1,N_2,...,N_k of N;
 7.             if k = Ø
                   then begin /* better solution found*/
 8.                   bound := cost(N);
 9.                   solution := N;
                   end
                   else /* store sons for later examination */
 1Ø.                  for i:=1 step 1 until k do
                          INSERT(PQ,N_i,h(N_i));
                end
          end
 11.    BB:=solution
      end;
```

performance of a branch and bound algorithm. The best-bound-first (bbf) search strategy [Lawler and Wood 1965; Fox, Lenstra, Rinnooy Kan, and Schrage 1978] chooses to apply the branching rule to that subset with the smallest lower bound. This strategy is realized by the heuristic function

$$h(S) = LB(S) \qquad (1)$$

where LB is the lower bound function used in a branch and bound algorithm. The relation $\leq_h$ is just the usual relation $\leq$ on the reals. In practice a priority queue is indeed the appropriate data structure for implementing a best-bound-first search. The ordered-depth-first (odf) search strategy applies the branching

rule to the least cost of the most recently split subsets and may be realized by

$$h(S) = (d(S), LB(S)) \tag{2a}$$

where d(S) = depth of the subset S in the tree generated by BB and the range of h is the set of ordered pairs. This heuristic function makes the priority queue simulate a stack each element of which is a priority queue, which is the way one would implement this search strategy in practice. The generation-order-depth-first (godf) search strategy applies the branching rule to the first generated of the subsets of a split set and can be realized by

$$h(S) = (d(S), i) \tag{2b}$$

for the i^{th} generated set. Again h produces an ordered pair. This heuristic function makes the priority queue simulate a stack whose elements are queues (or stacks; it does not matter). For both of these heuristic functions we define $(a,b) \leq_h (c,d)$ if and only if $a>c$ or ($a=c$ and $b \leq d$), i.e. subset S has higher priority $h(S) = (a,b)$ than subset T where $h(T) = (c,d)$ if and only if either S is deeper in the tree or S and T have the same depth but the lower bound on S is less than the lower bound on T. The ordered-breadth-first (obf) search strategy chooses to examine the least cost of the subsets which has the smallest depth in the tree generated by BB. A particular heuristic function realization is

$$h(S) = (d(S), LB(S)) \tag{3a}$$

as for depth-first search. In practice an ordered-breadth-first search is implemented using a separate priority queue for

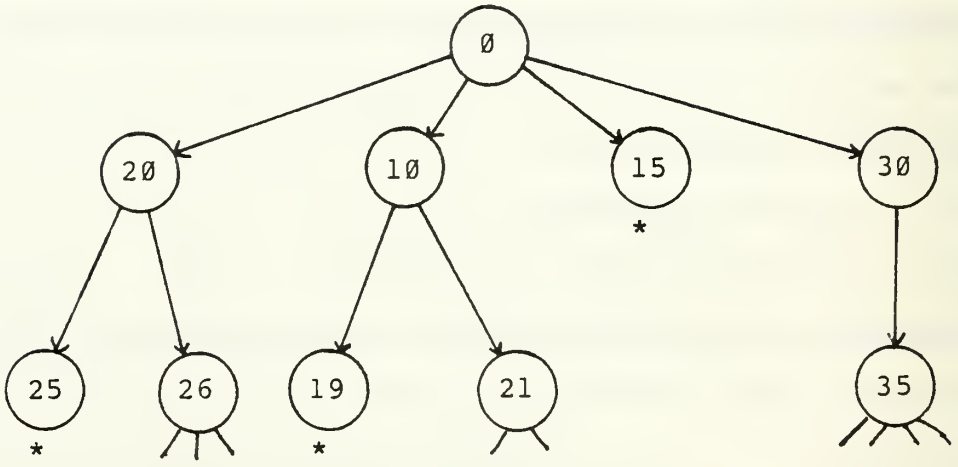
each level of the search tree so that the nodes on a given level can be extracted in order of increasing cost. Here we define $(a,b) \leq_h (c,d)$ if and only if $a<c$ or ($a=c$ and $b \leq d$). The generation-order-breadth-first (gobf) search strategy examines the subsets a level at a time in the order of their generation, and may be realized by

$$h(S) = (d(S),i) \tag{3b}$$

for the i^{th} generated node on level $d(S)$. In practice a generation-order-breadth-first search may be implemented using a single queue for storing nodes of the search tree. Note that according to the realizations given, both ordered-depth-first search and ordered-breadth-first search have local best-bound search components. E.g. in a breadth-first search a best-bound search is performed on the set of nodes that appear at a given depth. Figure 2.3 gives an example of a tree and the order in which each of the above search strategies examines the tree is given.

As an example of a branch and bound algorithm we will consider a subtour-elimination algorithm for solving traveling salesman problems. Subtour-elimination algorithms make use of a relaxation of the traveling salesman problem called the assignment problem (AP) which can be easily solved. The assignment problem comes from the problem of assigning n men to n jobs in a way which minimizes the cost of the assignment. We are given an nxn matrix $[c_{ij}]$ where c_{ij} is the cost of assigning man i to job j. The cost of an assignment is the sum of the costs of assigning each man to his job. The feasible set

Figure 2.3 A search tree and the order in which the search strategies realized by (1), (2a), (2b), (3a), and (3b) examine the nodes. A leaf is denoted by putting a star under a node. Lines under a node mean that if necessary the node could be split further.



bbf - <0, 10, 15, 19, 20, 21, 30>
godf - <0, 20, 25, 24, 15, 10, 19, 21, 30>
odf - <0, 10, 19, 21, 15, 20, 30>
gobf - <0, 20, 15, 10, 30, 25, 24, 19, 21>
obf - <0, 10, 15, 20, 30, 19, 21>

of an instance of the assignment problem can be formulated as a permutation of n objects. For example given the cost matrix

man \ job	1	2	3
1	1	3	7
2	4	2	5
3	3	2	6

the optimal assignment of men to jobs is: man 1 to job 1, man 2 to job 3, and man 3 to job 2, or (1)(2 3) in cycle notation. On the same cost matrix the optimal traveling salesman tour is (1 2 3). The assignment problem is a relaxation of the traveling salesman problem since a solution to the former is a permutation (composed of one or more cycles) whereas a solution to the latter must be a cyclic permutation (a permutation with just one cycle). The assignment problem is solveable in $O(n^3)$

time for an initial problem and $O(n^2)$ for subsequent modified versions of the initial problem [Bellmore and Malone 1971; Lawler 1976]. Subtour-elimination algorithms differ mainly in their choice of branching rule. The following branching rule was proposed by Shapiro [Shapiro 1966]:

Given cost matrix C, solve the assignment problem with respect to C. If the least cost solution, π, is cyclic, then we have extracted the least cost cyclic permutation over the feasible set of C, so there is no need to branch. If π is non-cyclic then pick one of its subcycles, say the smallest, and let this cycle be denoted $(i_1,i_2,\ldots,i_k)$. In the 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 feasible set is split as follows: In the j^{th} subset we force the node i_j to connect to a node not in the cycle $(i_1,i_2,\ldots,i_j)$ by setting the matrix entries

$$c_{i_j,i_1} = c_{i_j,i_2} = \ldots \quad c_{i_j,i_k} = \infty.$$

The lower bound function is simply the cost of the assignment problem solution. It is easily shown that this is a lower bound function. Condition 1 for a lower bound function (the lower bound function yields a lower bound on the cost of all objects in a given set), is satisfied since the assignment solution is by definition a lower bound on the cost of all permutations feasible with respect the cost matrix. Condition 2 (if $S_i \subseteq S_j$ then the lower bound on S_j is $\leq$ the lower bound on

S_i) is satisfied because the least cost permutation in a set S will have at least as small a cost as the least cost permutation in any subset of S. Condition 3 (if B(S)={S} then LB(S)=c(s^*) where s^* is the least cost object in S) is satisfied since a set S is not split if the assignment solution is also a traveling salesman solution. In this case the lower bound is just the cost of the least cost feasible object (a cyclic permutation) in S. There are a number of variations on the branching rule given in [Bellmore and Nemhauser 1971; Garfinkel 1973; Smith, Srinivasan, and Thompson 1977].

In all published versions the smallest subcycle of the assignment solution is chosen to guide the set splitting. This is justifiable on the general principle of tree searching that if possible it is wise to arrange the tree such that smaller branching factors are near the top of the tree and larger branching factors are deeper in the tree. The reason behind this principle is that if a node is pruned near the top of such a tree, there is a relatively larger reduction in the size of the feasible space due to the fact that the feasible space has been split fewer times by the smaller branching factor [Reingold, Neivergelt and Deo 1977, pages 111-112]. The choice of the smallest subcycle is good from another point of view. Bellmore and Malone [Bellmore and Malone 1971] have shown that this choice maximizes the reduction in the the number of feasible noncyclic solutions.

The subtour-elimination approach first appeared in [East-

man 1957] and was subsequently developed by [Shapiro 1966; Bellmore and Malone 1971; Smith, Srinivasan, and Thompson 1977].

Some Properties of Branch and Bound Algorithms

Efforts have been made to devise a formalism general enough to cover the diverse applications of the branch and bound procedure [Lawler and Wood 1966; Mitten 1970; Rinnooy Kan 1974, 1976; Kohler and Steiglitz 1974; Ibaraki 1976, 1978]. These formalisms have been used to prove correctness and termination properties and also to investigate theoretically the effects of various choices of parameters on performance. Although the theorems in this section are not essentially new the proofs are new in order to cover our different definitions and assumptions.

The first proposition allows us to assume that the heuristic realizations of the search strategies considered above are unambiguous.

Proposition 2.1: The best-bound-first, depth-first (both ordered and generation-order), and breadth-first (both ordered and generation-order) search strategies can be realized by unambiguous heuristic functions.

Proof: We can show that (1), (2a), and (3a) are unambiguous if the lower bound function LB satisfies the following con-

dition: for any S, $T \subseteq FS$ if $S \neq T$ then $LB(S) \neq LB(T)$. If for a particular problem and branch and bound algorithm LB does not satisfy this condition then a new lower bound function can be constructed at runtime as follows: LB'(S) = (LB(S),k) where S is the k^{th} distinct subset of FS with cost LB(S) examined to the current point in the search. With regards to the relation $\leq$ used in line 5 of figure 2.2, define $(a,b)<(c,d)$ if and only if $a<c$ or $(a=c$ and $b<d)$. Clearly LB' generates distinct bounds on distinct subsets. The heuristic function for best-bound-first search (1) h(S)=LB'(S) is unambiguous since all values of h for distinct subsets are distinct. The same reasoning holds for (2) and (3) since the range of h incorporates the lower bound LB'.

Let us now consider generation-order-depth-first search. Suppose there is a tree such that at some time during a generation-order-depth-first search there are distinct subsets S_1 and S_2 in memory with the same heuristic value $h(S_1) = h(S_2)$. Thus $d(S_1) = d(S_2)$ and $i_{S_1} = i_{S_2}$ where d(S) = depth of S and i_S is the generation number of S. S_1 and S_2 clearly cannot have the same parent set otherwise they would not have the same generation number. Suppose now without loss of generality that Sl was generated prior to S2. Let P2 denote the parent of S2. We have d(P2) = d(S2)-1. Since Sl is generated prior to S2 and there is a time when both Sl and S2 are in memory simultaneously, Sl must be on the queue when P2 is split to form S2 and its sibling sets. But P2 cannot be chosen over Sl for branching because $d(P2)<d(Sl)$ and therefore $h(P2)>h(Sl)$. This

contradiction shows that there cannot be two distinct sets on the queue with the same heuristic value under generation-order-depth-first search. A gobf is unambiguous because no two sets on a given level can be assigned the same generation number, so all assigned priorities are distinct. QED

The following sequence of definitions together with proposition 2.1 and lemma 2.1 lead up to a theorem regarding the conditions under which BB will find an optimal solution to an instance of a combinatorial minimization problem. Let A denote the sequence of nodes examined by $BB'(N_\emptyset,\infty)$ where BB' is BB with the test of statement 5 replaced by the value TRUE. Thus A is the order in which BB examines the entire tree if no cutoffs are performed. The possibility that the tree is infinite shall present no difficulties in defining A because we are only interested in the relative ordering of the nodes. In a similar way define A^b to be the sequence of nodes examined by $BB(N_\emptyset,b)$, i.e. BB given an initial bound of b. Note that A is not necessarily the same as A^∞ since cutoffs are made in the latter sequence. Lastly define B^b to be a sequence of values (bounds) associated with the nodes of A as follows: for each $i>\emptyset$, if node A_i is in A^b (i.e. A_i is examined by $BB(N_\emptyset,b)$), then B_i^b is the value of the variable bound in statement 5 at the time that A_i is examined by $BB(N_\emptyset,b)$. Otherwise $B_i^b=B_{i-1}^b$. Note that $B_\emptyset^b$ = b. As an example of these definitions consider the action of BB under a generation-order-depth-first search strategy on the tree of figure 2.3, we find

$A = \langle 0, 20, 25, 26, \ldots, 10, 19, 21, \ldots, 15, 30, 35 \rangle$

$A^{\infty} = \langle 0, 20, 25, 26, \quad 10, 19, 21, \quad 15, 30 \rangle$

$B^{\infty} = \langle \infty, \infty, \infty, 25, \ldots, 25, 25, 19, \ldots, 19, 15, 15 \rangle$

$A^{19} = \langle 0, 20, \quad 10, 19, 21, \quad 15, 30 \rangle$

$B^{19} = \langle 19, 19, 19, 19, \ldots, 19, 19, 19, \ldots, 19, 15, 15 \rangle$

Proposition 2.2 For all b, A^b is a subsequence of A.

Proof: Since A^b contains a subset of A it remains to be shown that the order of nodes in A is preserved in A^b. Assume the contrary. Let s be the first element of A^b which is out of order with respect to the ordering of A; so $A=\langle \ldots s, \ldots, r, \ldots \rangle$ $A^b=\langle \ldots r, s, \ldots \rangle$. There are two cases to consider. Case 1: At the time that s is examined in A, r is in the queue. By our assumption that the heuristic functions are unambiguous $h(s)<h(r)$. But in A^b if r and s are in the queue when r is examined then $h(r)<h(s)$. Or if s is not in the queue when r is examined it is because r is the parent of s. Thus r must be examined before s in A. Either way we have a contradiction. Case 2: At the time that s is examined in A, an ancestor $\hat{r}$ of r is in the queue (see figure 2.4). $\hat{r}$ must be examined sometime between the time that s is examined and the time that r is examined. Again by the restriction on the heuristic functions $h(s)<h(\hat{r})$. Since $h(s)<h(\hat{r})$ it must be that s is not in the queue when $\hat{r}$ is examined in A^b. Thus an ancestor, $\hat{s}$, of s must be in the queue with $\hat{r}$ and $h(\hat{r})<h(\hat{s})$. This implies that $\hat{s}$ follows $\hat{r}$ in A^b. On the other hand, $\hat{s}$ must precede s and since

Figure 2.4. A and A^b in case 2 of proposition 2.2. The inferred state of the priority queue is given under s in A and $\hat{r}$ in A^b. The topmost entries have the highest priority.

$A=<...\hat{s},...,s,...,\hat{r},...,r,...>$ $\begin{bmatrix} s \\ \cdots \\ \hat{r} \end{bmatrix}$ $A^b = <...,\hat{r},...,\hat{s},...,r,s,...>$ $\begin{bmatrix} \hat{r} \\ \cdots \\ \hat{s} \end{bmatrix}$

$h(s)<h(\hat{r})$ $\hat{r}$ must follow s in A, thus $\hat{r}$ follows $\hat{s}$ in A. But this means that $\hat{r}$ is out of order contradicting our assumption that s is the first node in A^b out of order. QED

The following lemma asserts that exactly those nodes are examined in a branch and bound search wih finite initial bound whose parents have a cost within the current value (at examination time of the parent) of the bound.

Lemma 2.1: For any node M in A, except the initial node $N_\emptyset$, and for any finite integer b, M is examined by $BB(N_\emptyset,b)$ if and only if $LB(M')<B^b_{index(M')}$, where M'=parent(M) and index(N) is the index in A of node N. LB is the lower bound function.

proof: If part: We will show by contradiction that all nodes in A except $N_\emptyset$ satisfy this half of the lemma. Let M be the first node in A which is not examined by $BB(N_\emptyset,b)$, and whose parent, M', has cost $cost(M')<B^b_{index(M')}$. We can show that M' is examined by $BB(N_\emptyset,b)$ as follows: if $M'=N_\emptyset$ then we already know that M' is examined since the initial node is always examined by BB. Otherwise let M"=parent(M'). We can show that

$LB(M") < B^{b}_{index(M")}$. We have $LB(M") \leq LB(M')$ by condition 2 of the definition of a lower bound function, and $B^{b}_{index(M")} \geq B^{b}_{index(M')}$ since M" comes before M' in the node ordering of A and the sequence B^{b} is nonincreasing. Thus we have $LB(M") < LB(M') < B^{b}_{index(M')} \leq B^{b}_{index(M")}$. Since we have assumed that M is the first unexamined node in A such that $LB(M') < B^{b}_{index(M')}$, it must be that M' is examined by $BB(N_{\emptyset},b)$. When M' is examined we have $LB(M') < B^{b}_{index(M')}$, i.e., the lower bound of M' is within the current value of the program variable bound, thus the test of line 5 in Figure 2.2 evaluates to true and the sons of M' (including M) are inserted in the priority queue for later examination. For any finite initial bound b, there are only a finite number of nodes with a lower bound less than b and since any node can insert a finite number of sons in the queue, there are only a finite number of nodes inserted in the queue during the execution of $BB(N_{\emptyset},b)$. This means that after a finite time the node M must be examined. This statement contradicts our assumption that M is not examined by BB, so our assumption that there is a node that is not examined under the condition of the lemma must be false.

For the only if part: if M is examined by $BB(N_{\emptyset},b)$ then by definition M was inserted in the priority queue at some time. This could only be if, when the parent of M, M', was examined, the test of line 5 in Figure 2.2 evaluated to true, i.e., $LB(M') < bound = B^{b}_{index(M')}$. QED

It can now be shown under what conditions BB will solve

an instance of a combinatorial minimization problem.

Theorem 2.1: For finite $b>c^*$ where c^* is the cost of a least cost object in FS, $BB(N_\emptyset,b)$ will find an optimal solution of cost c^* in a finite amount of time.

If c^* is the cost of a least cost object in FS then for all nodes in A, $B_i^b \geq c^*$ where i ranges over the nodes of A (we have $B_\emptyset^b => c^*$, and there is no object in FS which could lower the bound below c^*). Let s^* be the first node in A which represents a subset of FS from which the branching rule extracts a least cost object of cost c^*. The parent of s^* clearly has a lower bound$<c^*$ so by lemma 2.1, s^* is examined by $BB(N_\emptyset,b)$. Thus $BB(N_\emptyset,b)$ finds an optimal solution. Again since only a finite number of nodes are involved when b is finite BB terminates in a finite amount of time. QED

We conclude this chapter with the following theorem which asserts that it is worthwhile to find as tight an initial bound as possible on the cost of the optimal solution in order to minimize the amount of work necessary to find the optimal solution.

Theorem 2.2: For a particular instance of a combinatorial minimization problem p=(FS,cost) where cost$\in COST_n$ in class C_n, let ET(h,p,b) be the number of subsets examined by $BB(N_\emptyset,b)$ using the search strategy realized by h. For all h and all b,b'

such that $b'>b\geq\emptyset$, $ET(h,p,b)\leq ET(h,p,b')$.

Proof: The theorem follows easily from the following lemma.

Lemma 2.2: For all i and $b'>b$, $B_i^b \leq B_i^{b'}$.

Proof by induction on i. For $i=\emptyset$, $B_\emptyset^b=b$ and $B_\emptyset^{b'}=b'$ so the lemma holds. Next assume $B_{i-1}^b \leq B_{i-1}^{b'}$ for some $i>1$. B_i^b will be changed from the value B_{i-1}^b only if $BB(N_\emptyset,b)$ examines A_{i-1} and it represents a solution which changes the bound. If $B_i^b > B_i^{b'}$ it must be because $B_i^{b'} = cost(A_{i-1})$ but B_i^b remains unchanged. Let A_j be the parent of A_{i-1}. A_{i-1} is unexamined in $BB(N_\emptyset,b)$ if $cost(A_j)\geq B_j^b$ by lemma 2.1. But since A_{i-1} is examined by $BB(N_\emptyset,b')$ we have $cost(A_j)<B_j^{b'}$, thus $B_j^{b'}<B_j^b$ which contradicts our inductive assertion. So $B_i^b \leq B_i^{b'}$.

Let A_i be a subset examined by $BB(N_\emptyset,b)$, Let A_j denote the parent of A_i. By lemma 2.1, $cost(A_j)<B_j^b$ and $B_j^b \leq B_j^{b'}$ by lemma 2.2, so $cost(A_j)<B_j^{b'}$. Thus A_i is also examined by $BB(N_\emptyset,b')$ according to lemma 2.1. If every subset examined by $BB(N_\emptyset,b)$ is also examined by $BB(N_\emptyset,b')$ then $ET(h,p,b)\leq ET(h,p,b')$. QED

Chapter 3.

Computational Complexity and a Model of Branch and Bound Search Trees

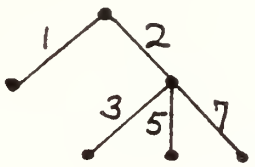
3.1 Random Problems and Random Trees

In the previous section it was noted that the branch and bound process generates a tree structure. In this chapter we use this abstraction to define a probabilistic class of trees which models the kind of tree structures that BB generates over the instances of a combinatorial minimization problem. Within this model then it makes sense to derive expressions for the expected time and space requirments of BB under various search strategies. The set of subsets of FS that are inserted in the priority queue during the execution of BB is called the search tree and 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 subsets in the queue at any time during the search. The time and space complexities of a given search by BB will sometimes be denoted by the variables N_T and N_S respectively. This definition of time complexity does not include the amount of time spent executing the branching rule or inserting nodes in the queue. A branching rule is a feature of a particular algorithm and little can be said about it on the level of abstraction aimed for

in this dissertation. We assume that these factors are relatively independent of the rest of the branch and bound process so that the product of the average branching time per subset, the average time spent inserting a node in the queue, and the total number of nodes inserted in the queue (the time complexity) is a reasonable approximation to the running time of a particular problem on a machine. Again though our goal is to compare the expected performance of various search strategies on a problem. On a given problem the branching time and node insertion time should factor out of this comparison leaving the size of the search tree as the essential measure of performance.

The question of interest is how can we model the behavior of BB on a random instance of a problem apart from the details of the problem. I.e, what features of a branch and bound search are relevant to branch and bound and what are problem dependent? First by the action of the branching rule a tree structure is generated, so BB is a tree searching algorithm. Secondly 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 expressing 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-labelled tree is a tree T=(N,A,C) where N is a set of nodes, A is a set of arcs, and C:A->Z (positive integers) is a cost function on the arcs of the tree. For exam-

Figure 3.1. An arc-labelled tree



ple see figure 3.1. In an arc-labelled 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-labelled tree domain. A probability function is assigned to the class of arc-labelled trees which should somehow correspond with a probability distribution on a combinatorial 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 experiment. Formally let P and Q be probability mass functions. It is assumed that P and Q satisfy the following properties:

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

The algorithm in figure 3.2 generates a random arc-labelled tree. Let RANDOM(F) be a random function which returns k with

probability F(k). This dynamic means of defining a random arc-labelled tree is easily implemented for experimental purposes on a machine.

This process is related to the well-known branching process [Harris 1963] which has applications to population growth, nuclear fission reactions, and particle cascades. The basic branching process is essentially the same as the process in figure 3.2 except that sprouting may be done in parallel and there is no arc-labelling. The initial node in a branching process is viewed as an individual who gives birth to k individuals with probability P(k), who in turn give birth to new individuals, and so on. The number of nodes at depth d in the generated tree is the random variable of interest and it is interpreted as the size of the population at time d. The theory of branching processes is concerned with the distribution and moments of the population size as a function of time, the probability of extinction (i.e., whether the tree is finite or infinite), and the behavior of the process in the case that the population does not die out. In contrast our concern here is with the behavior of the algorithm BB on a randomly generated tree. In general only a small finite portion of the tree will be searched by BB.

We will need to define a probability function on the set of arc-labelled trees. This can be accomplished as follows. The generation of a tree is viewed as a sequence of trials, where each execution of step 2 in figure 3.2 is a trial. Let n

Figure 3.2. Generating a random arc-labelled 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.

denote the number of sons generated in a random trial and let $c_1, c_2, \ldots, c_n$ denote the arc costs assigned to the arcs. The probability of the outcome of a trial then is $P(n)Q(c_1)Q(c_2)\ldots Q(c_n)$. Clearly if we sum over all possible outcomes of a trial, the probabilities sum to 1,

$$\sum_{n=\emptyset}^{\infty} P(n) \sum_{c_1=1}^{\infty} Q(c_1) \ldots \sum_{c_n=1}^{\infty} = 1.$$

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 $\langle g_\emptyset, g_1, g_2, \ldots \rangle$, where g_i is the probability of the particular outcome of the i^{th} trial. Let us call the product $g_\emptyset g_1 \ldots g_i$ the i^{th} 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 i^{th} partial probability. For example, the probability of the arc-labelled tree in Figure 3.1 is P(2)*Q(1)*Q(2)*P(Ø)*P(3)*Q(3)*Q(5)*Q(7)*P(Ø)*P(Ø)*P(Ø). It is our special assumption that each trial is independent of all other trials that enables us to take the product of the proba-

bilities of the individual trials as the probability of the tree. A more formal approach to this probability function over the set of arc-labelled trees can be based on the measure-theoretic treatment of trees generated by a branching process found in [Mode 1971, pg. 3-6].

If the tree generated by our process is infinite then the limit of the partial probabilities will usually go to zero since we are considering the product of numbers between 0 and 1. Although the probability of generating a particular infinite tree is usually zero, it can be shown that the probability that a randomly generated tree is infinite is nonzero for many probability functions P. This fact is a basic result of the theory of branching processes [Feller 1963, pg. 7] and may be stated more precisely as follows: Let $\overline{P}$ denote the mean of P. If $\overline{P} \leq 1$ then a randomly generated tree is finite with probability 1. If $\overline{P} > 1$ then a randomly generated tree is infinite with probability ξ, where ξ is the least positive fixed point of the generating function for P: $\xi = p(\xi)$ where

$$p(s) = \sum_{k=0}^{\infty} P(k)s^k.$$

A randomly generated tree is finite with probability $1-\xi$. By an infinite tree we mean not only a tree with unbounded depth but also that the number of nodes at depth d grows unboundedly in d. It turns out that the probability that a random tree has unbounded depth but a bounded nonzero number of nodes on all levels is zero for all P.

Let sons(N) denote the number of sons of the node N. The arc-labelled tree T is in the class of (P,Q)-trees if and only if $P(sons(N))>0$ for all $n \in N$ and $Q(C(a))>0$ for all $a \in A$. E.g., if a node in T has 11 sons but $P(11)=0$ then T is not a (P,Q)-tree.

The remainder of this dissertation is concerned with the expected performance of BB on the class of (P,Q)-trees for arbitrary P and Q. All theorems about (P,Q)-trees are implicitly quantified over all probability mass functions P and Q though it may not be stated. It is important to ask how successful this transfer is of the notion of random problem instance to a random (P,Q)-tree. Can we predict (through careful choice of P and Q) the expected performance of BB on a combinatorial minimization problem by finding the expected performance of BB on the class of (P,Q)-trees?

The key asumption in this model is the independence of each application of the branching rule and the independence of each assignment of arc costs. It might be expected however that the degree of a node depends somewhat on the depth in the tree. In particular for finite trees the probability that a node has zero sons should go to one with increasing depth. That tnese observations are so for the traveling salesman problem is borne out by table 2 in chapter 7.

In defense of this model it may be noted that this is perhaps the simplest possible model of branch and bound trees and should prove more amenable to analysis than a more complex

model. With regards to the independence assumption, 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 chapter 7 that the theory of (P,Q)-trees can be applied with good predictive power to a branch and bound algorithm for solving traveling salesman problems, and we conjecture that the model is applicable to at least those branch and bound algorithms which employ a relaxation procedure.

Some notation follows which will be needed. For a random variable x and a probability mass function p(x), the expected value and variance of x are computed by

$$E(x) = \sum_{x} x\, p(x)$$

$$\sigma_x^2 = E(x^2) - E(x)^2.$$

The first and second moments of P will be denoted $\overline{P}$ and $\bar{\bar{P}}$ respectively, i.e.,

$$\overline{P} = \sum_{k \geq \emptyset} kP(k) \quad \text{and}$$

$$\bar{\bar{P}} = \sum_{k \geq \emptyset} k^2 P(k).$$

3.2 Properties of a Class of (P,Q)-trees.

Before studying the behavior of BB on (P,Q)-trees it will be useful to develop expressions for some important properties of a class of (P,Q)-trees. For example what is the expected

path length of a randomly picked path in a randomly picked tree? The probability that a node is a leaf is $P(\emptyset)$ and the probability that a node has some sons is $1-P(\emptyset)$. A branch of length k then has probability $(1-P(\emptyset))^k P(\emptyset)$, a geometric distribution. The expected path length is

$$\sum_{k=\emptyset}^{\infty} k(1-P(\emptyset))^k P(\emptyset) = (1-P(\emptyset))/P(\emptyset). \qquad (1)$$

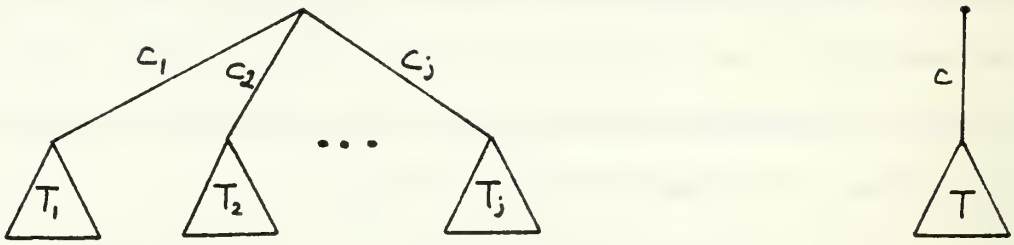
A more difficult question concerns the distribution of least cost leaves over the class of (P,Q)-trees. Let opt(T) denote the cost of the least cost leaf in an arc-labelled tree T. Let $\hat{O}(i)$ denote the probability that opt(T)=i in a random (P,Q)-tree T. $\hat{O}$ is defined on the nonnegative integers since the cost of any leaf in a (P,Q)-tree is a nonnegative integer by definition. A recurrence relation for $\hat{O}$ can be formulated by equating two expressions for the probability that opt(T)>i in a random (P,Q)-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 $\hat{O}(\emptyset) = P(\emptyset)$. One expression for the probability that opt(T)>i is

$$1 - \sum_{k=\emptyset}^{i} \hat{O}(k). \qquad (2)$$

Next consider the treetop shown in Figure 3.3a. The subtrees $T_1, T_2, \ldots, T_j$ are themselves random (P,Q)-trees. The probability that $opt(T'_k)>i$ where T'_k is the kth subtree plus the arc from the root as in figure 3.3b is

$$1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c). \qquad (3)$$

Figure 3.3a. A Tree-top. 3.3b. A Branch



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 tree-top of Figure 3.3a has j branches and opt(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>Ø the probability that opt(T)>i in a random (P,Q)-tree is

$$\sum_{j=1}^{\infty} P(j)\left[1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}. \qquad (4)$$

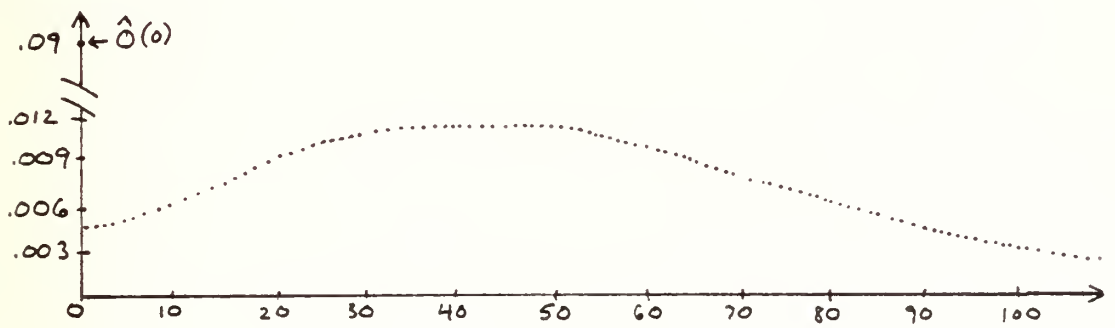
The case j=Ø is not included in this expression because then opt(T) = Ø. Finally expressions (2) and (4) can be equated:

$$1 - \sum_{k=\emptyset}^{i} \hat{O}(k) = \sum_{j=1}^{\infty} P(j)\left[1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)\right]^{j}. \qquad (5)$$

This is a recurrence relation since $\hat{O}(i)$ appears on the left but only the values $\hat{O}(\emptyset)$, $\hat{O}(1)$, ..., $\hat{O}(i-1)$ appear on the right

for $i \geq 1$. In the appendix 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. Empirical data on uniformly distributed P and Q suggests that $\hat{O}(n)$ is asymptotic to d^n as $n \to \infty$ for some constant d that depends on P and Q. Figure 3.4 shows some of $\hat{O}$ for the class of (P_{10}, Q_{100})-trees where $P_{10}(k) = 1/11$ if and only if $0 \leq k \leq 10$, and $Q_{100}(c) = 1/100$ if and only if $1 \leq c \leq 100$.

Figure 3.4. $\hat{O}(i)$ for (P_{10}, Q_{100})-trees.



Let dep(T) denote the least depth at which a least cost leaf may be found in an arc-labelled tree T. A generalization of the function $\hat{O}(i)$ is the function d(i,k) = probability that opt(T)=i and dep(T)=k in a random (P,Q)-tree T. In a manner similar to the derivation of (13) above, a recurrence relation can be formulated for d(i,k) by equating two expressions for the probability that opt(T)>i or (opt(T)=i and dep(T)$\geq$k).

$$1 - \sum_{j=0}^{i-1} \sum_{m=0}^{j} d(j,m) - \sum_{m=0}^{k} d(i,m)$$

$$= \sum_{j=1}^{\infty} P(j)[1 - \sum_{c=1}^{i} Q(c)d(0,0) - \sum_{c=1}^{i-2} Q(c) \sum_{m=1}^{i-c-1} \sum_{n=1}^{m} d(m,n)$$

$$- \sum_{c=1}^{i-1} Q(c) \sum_{m=1}^{k-1} d(i-c,m)]^{j} \qquad (6)$$

where $d(0,0) = P(0)$

$d(i,m) = 0$ for $m>i$

$d(i,0) = 0$ for $i>0$

Note that the left hand side of (6) has the term $d(i,m)$ whereas the righthand side uses only the terms $d(j,k)$ for $j<i$ or ($j=i$ and $k<m$).

By taking marginal sums of $d(i,m)$ we obtain two important functions concerning the class of (P,Q)-trees. First we can derive the recurrence relation (4) for $\hat{O}(i)$ again since:

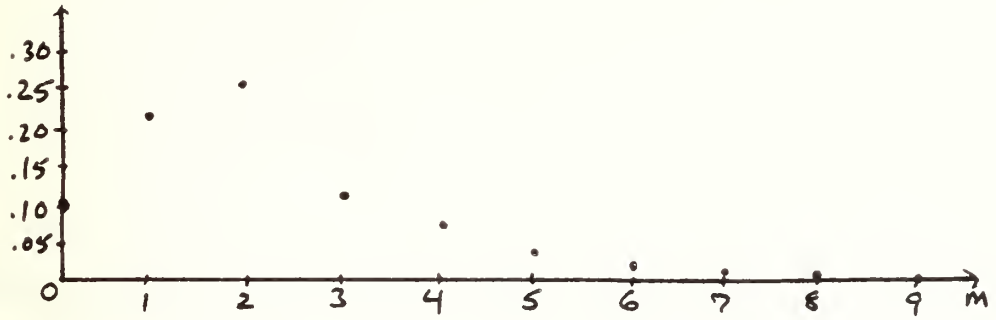
$$\hat{O}(i) = \sum_{l=0}^{\infty} d(i,m) = \sum_{m=1}^{i} d(i,m) \quad \text{for } i>0 \qquad (7)$$

Secondly, let DEP(m) be the probability that dep(T)=m in a randomly generated (P,Q)-tree T. This function is given by

$$DEP(m) = \sum_{i=m}^{\infty} d(i,m) \qquad (8)$$

Figure 3.5 shows an example distribution of DEP(m). Note that DEP(m) quickly approaches zero as m increases.

Figure 3.5. DEP(m) for the class of (P_{10},Q_{100})-trees.



Chapter 4.

Heuristic Search Strategies.

Subsequent chapters will investigate some particular well known search strategies but it seems appropriate to first look at search strategies in general. In chapter 2 the function ET(h,p,b) was defined as the time complexity for solving the problem instance p when BB uses the search strategy realized by the heuristic function h and is given an initial bound of b. Define the expected time complexity of a heuristic search of a random (P,Q)-tree as

$$ET_h(b) = \sum_t Pr(t)ET(h,t,b) \qquad (1)$$

where t varies over all (P,Q)-trees and Pr(t) is the probability of t as defined in the previous chapter. Of particular interest for comparison purposes in this thesis will be the limit as b goes to ∞ of $ET_h(b)$, denoted by $E(N_T)$ or $ET_h(\infty)$.

The time complexity of a branch and bound search may be viewed as a random sum of independent variables. The i^{th} such variable is the number of sons inserted on the queue by the i^{th} explored node. Let $G_h(k)$ denote the probability that exactly k nodes are explored during the search of a random (P,Q)-tree under search strategy h. If we let X_i denote the number of sons that the i^{th} explored node has then the size of the search

tree, denoted by the random variable N_T, is a random sum $1 + X_1 + X_2 + \ldots + X_k$ where each X_i is distributed according to P and k is distributed according to G_h. See section 3.1 for the definitions of $\overline{F}$ and $\bar{\bar{F}}$.

Theorem 4.1. Let $N_T = 1 + X_1 + X_2 + \ldots + X_k$ be a random sum where each variable X_i is distributed according to P and k is a random variable distributed according to G_h, then

$$E(N_T) = 1 + \overline{PG}_h \tag{2}$$

$$\sigma^2_{N_T} = \bar{\bar{P}}\overline{G} + \overline{P}^2(\bar{\bar{G}}_h - \overline{G}_h - \overline{G}_h^2). \tag{3}$$

Proof: Let p(z) and g(z) be the generating functions for P and G_h respectively, i.e.

$$g(z) = \sum_{n=\emptyset}^{\infty} G_h(n)z^n, \qquad p(z) = \sum_{n=\emptyset}^{\infty} P(n)z^n.$$

It is a well-known result (c.f. Feller 1959 pg. 286-287) that the generating function for the random sum $X_1 + X_2 + \ldots + X_k$ is g(p(z)). It can also be shown (Feller 1959, pg. 265-266) that for a variable x distributed according to F(x) with generating function f(z),

$$f(1) = 1 \tag{4}$$

$$E(x) = f'(1) \text{(first derivative of f)}, \tag{5}$$

Let f" denote the second derivative of f, then

$$f''(1) = \bar{\bar{F}} - \overline{F} \tag{6}$$

$$\sigma^2_x = f''(1) + f'(1) - f'(1)^2. \tag{7}$$

From these relations it is straightforward to derive the rela-

tions of the theorem. In what follows we use the symbol D_z in the usual way as the derivative with respect to z.

$$E(N_T-1) = D_z\, g(p(z))|_{z=1} \qquad \text{by (5)},$$
$$= g'(p(z))p'(z)|_{z=1}$$
$$= g'(p(1))p'(1)$$
$$= g'(1)p'(1)$$
$$= \overline{GP}.$$

Thus we have shown

$$E(N_T) = 1 + \overline{GP}.$$

The variance can also be derived straightforwardly.

$$D_z^2\, g(p(z))|_{z=1} = D_z\, g'(p(z))p'(z)|_{z=1}$$
$$= g''(p(z))p'(z)p'(z) + p''(z)g'(p(z))|_{z=1}$$
$$= g''(1)p'(1)^2 + p''(1)g'(1)$$
$$= (\bar{\bar{G}} - \bar{G})\bar{P}^2 + (\bar{\bar{P}} - \bar{P})\bar{G}. \qquad \text{by (6)}$$

Thus

$$\sigma^2_{N_T-1} = D_z^2 g(p(z))|_{z=1} + D_z g(p(z))|_{z=1} - (D_z g(p(z))|_{z=1})^2$$
by (7)
$$= (\bar{\bar{G}}\bar{P}^2 - \overline{GP}^2 + \bar{\bar{P}}\bar{G} - \overline{PG}) + \overline{PG} - \bar{P}^2\bar{G}^2.$$
$$= \bar{P}^2(\bar{\bar{G}} - \bar{G} - \bar{G}^2) + \bar{\bar{P}}\bar{G}.$$

Now since $\sigma^2_{N_T-1} = \sigma^2_{N_T}$, we have

$$\sigma^2_{N_T} = \bar{P}^2(\bar{\bar{G}} - \bar{G} - \bar{G}^2) + \bar{\bar{P}}\bar{G}. \quad \text{QED}$$

We immediately obtain the following corollary.

Corollary 4.1: $E(N_T)$ exists if and only if $\overline{P}$ and $\overline{G}$ exist. $\sigma^2_{N_T}$ exists if and only if $\overline{P}$, $\overline{G}$, $\overline{\overline{P}}$, and $\overline{\overline{G}}$ exist.

Since the function P is assumed given, the main task in the analysis of a heuristic search strategy is to find a formulation for $G_h(k)$. Lower bounds can be found on the means of N_T and N_S for any h.

Theorem 4.2. For all exact heuristic functions h,

$$E_h(N_T) \geq 1 + \overline{P}/P(\emptyset),$$

$$E_h(N_S) \geq 1 + (\overline{P}-1)/P(\emptyset).$$

Proof: By lemma 2.1 any exact search strategy must explore the least cost leaf s^* and all the nonleaf nodes n such that $LB(n)<c(s^*)$. Let $\hat{h}$ be a heuristic function for a search strategy which explores those nodes and no others. If we imagine the nodes of a tree laid out in a sequence in order of increasing cost then $\hat{h}$ explores just those nodes up to the first leaf in the sequence. The probability that $\hat{h}$ explores k nonleaf nodes is given by

$$G_{\hat{h}}(k) = (1-P(\emptyset))^k P(\emptyset) \tag{8}$$

i.e, the first k nodes in the sorted sequence are nonterminal (each with probability $1-P(\emptyset)$) and exactly one leaf s^* is explored (with probability $P(\emptyset)$). We have

$$\overline{G}_{\hat{h}} = \sum_{k=\emptyset}^{\infty} k(1-P(\emptyset))^k P(\emptyset) = (1-P(\emptyset))/P(\emptyset) \tag{9}$$

Also each nonleaf has j sons with probability $P'(j) = P(j)/1-P(\emptyset)$ for $j \geq 1$, thus

$$\overline{P}' = \sum_{j=1}^{\infty} jP(j)/(1-P(\emptyset)) = \overline{P}/(1-P(\emptyset)) \qquad (10)$$

By theorem 4.1,

$$E_h^{\wedge}(N_T) = 1 + (1-P(\emptyset))/P(\emptyset) * \overline{P}/(1-P(\emptyset))$$

$$= 1 + \overline{P}/P(\emptyset).$$

It follows that for any exact heuristic function h,

$$E_h(N_T) \geq 1 + \overline{P}/P(\emptyset).$$

The space complexity of a random (P,Q)-tree under any search strategy is bounded below by the number of nodes on the queue when the first leaf is found by the search. Again N_S is a random sum, but a random sum of random variables which are slightly different from the random variables X_i in N_T. Let $G'_h(k)$ be the probability that k nonleaves are explored before the first leaf is found. $G'_h(k)$ is the same as $G_h^{\wedge}(k)$ formulated above, so

$$\overline{G}'_h = (1-P(\emptyset))/P(\emptyset).$$

During the exploration of the i^{th} node, the node itself is removed from the queue and X_i nodes are added, thus the net increase to the queue size is X_i-1, denoted X'_i. The random variables X'_i are distributed according to P' where $P'(x) = P(x+1)/(1-P(\emptyset))$. We have

$$\overline{P}' = \sum_{j=\emptyset}^{\infty} jP'(j+1) = \sum_{j=\emptyset}^{\infty} jP(j)/(1-P(\emptyset))$$

$$= \sum_{j=0}^{\infty} (j+1)P(j+1) - \sum_{j=0}^{\infty} P(j+1)$$

$$= \frac{\bar{P} - (1-P(0))}{1-P(0)}$$

Therefore

$E(N_S) \geq \bar{G}'_h \bar{P}'$ (mean of the random sum $X'_1 + X'_2 + \ldots + X'_k$).

$$= \frac{\bar{P} - (1-P(0))}{1-P(0)} \frac{1-P(0)}{P(0)}$$

$$= 1 + (\bar{P}-1)/P(0). \quad \text{QED}$$

4.2 Techniques for analyzing a heuristic search strategy.

The state of a branch and bound search process at the beginning of statement 3 of BB (see figure 2.2) can be described by the state of the priority queue and the value of the bound. An actual branch and bound process may be described by a sequence of such states. If we can give the probability that a random process in state S_i will be in state S_j at the next execution of statement 3 of BB given only that the current state is S_i, then the set of all such states (including an initial state) and the probabilities of the transitions between states defines a markov chain. Formally let $\{S_\emptyset, S_1, S_2, \ldots\} \cup \{F_\emptyset, F_1, \ldots\}$ denote the possible states of a search where $S_\emptyset = \langle b_\emptyset; N_\emptyset \rangle$ is the initial state, and $S_i = \langle b; n_1, n_2, \ldots, n_k \rangle$ where b is a value of the bound and the nodes $n_1, n_2, \ldots, n_k$ are on the queue in that order, i.e., $h(n_1) < h(n_2) < \ldots < h(n_k)$. The final states are denoted by $F_b = \langle b; \rangle$ where b is the value of the bound and the priority queue is empty (an empty priority queue

terminates BB). Let r_{ij} denote the probability that the process in state i will make a transition from state i to state j. Let R denote the (infinite) transition probability matrix $[r_{ij}]$. Suppose we wish to describe the behavior of BB under search strategy h and given an initial bound of $b_\emptyset$. The initial state is $\langle b_\emptyset ; N_\emptyset \rangle$. The transition probabilities may be described as follows.

The transitions

$$\langle b; n_1, n_2, \ldots, n_k \rangle \rightarrow \langle b; n_2, \ldots, n_k \rangle \qquad (11)$$

occur with probability 1 for all states in which $b \leq c(n_1)$. These transitions reflect the act of pruning the subtree below n_1 as effectively happens when statement 5 in figure 2.2 is false. The transitions

$$\langle b; n_1, n_2, \ldots, n_k \rangle \rightarrow \langle c(n_1); n_2, \ldots, n_k \rangle \qquad (12)$$

occur with probability $P(\emptyset)$ for all states in which $b > c(n_1)$. These transitions reflect the action taken by BB on states for which statement 5 is true and statement 7 is true. (a leaf is found which improves the value of the bound). The transitions

$$\langle b; n_1, n_2, \ldots, n_k \rangle \rightarrow \langle b; m_1, m_2, \ldots, m_{k-1+j} \rangle \qquad (13)$$

occur with probability $P(j)Q(c_1)Q(c_2)\ldots Q(c_j)$ where

$\{m_1, m_2, \ldots, m_{k-1+j}\} = \{n_1, n_2, \ldots, n_k, n_1+c_1, n_1+c_2, \ldots, n_1+c_j\}$

and $h(m_1) < h(m_2) < \ldots h(m_{k-1+j})$.

These transitions reflect the action taken by BB on states for

which statement 5 in figure 2.2 is true but statement 7 is false; (the sons of a node are added to the queue for later exploration).

Since a node is removed from the queue at each transition the number of transitions from the start state to a final state, called the first passage time for a random process, is the number of nodes inserted on the queue during the search; i.e., the time complexity. There are well-known methods for finding first passage times and their means [Parzen 1973], but they are not of much help when the transition matrix is infinite. A sequence of states $\{S_0, S_1, \ldots, S_n\}$ is <u>realizable</u> if S_0 is the initial state, S_n is a final state, and for $0 \leq i < n$ $r_{S_i S_{i+1}} > 0$. The set of all realizable sequences defines the sample space on which our random variables N_T for the time complexity, and N_S for the space complexity are defined. The probability of a realizable sequence is defined to be

$$\prod_{i=0}^{n-1} r_{S_i S_{i+1}}.$$

Insight is needed into the nature of a particular search strategy in order to coarsen this sample space into appropriate events such that an expression for the expected complexities can be derived. Theorem 4.1 offers some help in this direction by defining the event E_n as the set of all realizable sequences in which exactly n transitions of the types (12) and (13) appear. We have defined the probability of event E_n as $G_h(n)$. Since P is given and it is assumed that $\overline{P}$ can be found, the

problem of finding $E(N_T)$ reduces to finding a reasonable expression for $\bar{G}_h$, perhaps by first finding $G_h(n)$.

Complementing this general approach to finding $E(N_T)$ and $E(N_S)$ for a heuristic search strategy, there is a more ad-hoc method. In this approach we observe how the data structure which is normally used to implement the search strategy is affected by searching a random (P,Q)-tree. Special properties of these data structures may help in the analysis. For example, a depth-first search is usually implemented using a stack. The close relationship between stacks and the implementation of recursion suggests that a recurrence relation may be the best way to describe an algorithm employing a stack.

Chapter 5.

The Best-Bound-First Search Strategy

The best-bound-first search strategy as realized by 3-(1) chooses to explore the least cost unexplored node on the priority queue. The following theorem gives a characteristic property of this search strategy.

Theorem 5.1. The first leaf explored in an arc-labelled tree by BB under the best-bound-first search strategy is optimal.

Proof: Best-bound-first explores the nodes of a tree in order of increasing cost. The first leaf s^* which is explored then is optimal since all nodes (including leaves) which could be explored subsequently have at least as great a cost as s^*. QED

As a result of theorem 5.1 it is not neccesary to explore all stored nodes in a best-bound-first search. By the nature of the heuristic function all nodes on the queue when the first leaf has been found have a cost greater than or equal to the cost of the leaf. Therefore there is no need to explore any further nodes and the search may terminate. More generally, whenever a search strategy has a best-bound-first component (e.g. in ordered-depth-first, ordered-breadth-first) as soon as

a leaf is found or a node with greater cost than the bound, then no more nodes need be examined from the priority queue.

Theorem 5.2. The expected time and space complexities of a best- bound-first search of a random (P,Q)-tree are,

$$E_{bbf}(N_T) = 1 + \overline{P}/P(\emptyset) \tag{1}$$

$$E_{bbf}(N_S) = 1 + (\overline{P}-1)/P(\emptyset) \tag{2}$$

Proof: Since the best-bound-first search strategy examines nodes in order of increasing cost, it is a special case of the heuristic function $\hat{h}$ analyzed in theorem 4.2. Therefore the results derived for $\hat{h}$ also hold for best-bound-first search. QED

As a consequence of theorem 5.2 the best-bound-first search strategy is optimal both in terms of time and space within our model. Using theorems 4.1 and 4.2 it is possible to derive expressions for the variances of N_T and N_S also.

Theorem 5.3. The variances of the performance of a best-bound-first search on a random (P,Q)-tree are

$$\sigma^2_{N_T} = \overline{P}^2/P(\emptyset)^2 + \overline{\overline{P}}/P(\emptyset) \tag{3}$$

$$\sigma^2_{N_S} = \overline{(P-1)}^2/P(\emptyset)^2 + (\overline{\overline{P}}-1)/P(\emptyset) \tag{4}$$

Proof: The random variable N_T is a random sum $1 + X_1 + X_2$

+ where $X_i \geq 1$ has the probability

$$P'(X_i) = P(X_i)/(1-P(\emptyset)),$$

and the probability that there are k terms (the number of non-terminal nodes) in the sum is

$$G(k) = P(\emptyset)(1-P(\emptyset))^k.$$

as in theorem 4.2. The variance of N_T is

$$\sigma^2_{N_T} = \overline{P}'^2(\overline{\overline{G}}-\overline{G}-\overline{G}^2) + \overline{\overline{P}}'\overline{G}. \qquad (5)$$

by theorem 4.1. We have

$$\overline{P}' = \sum_{X=1}^{\infty} X\frac{P(X)}{1-P(\emptyset)} = \frac{\overline{P}}{1-P(\emptyset)},$$

$$\overline{\overline{P}}' = \sum_{X=1}^{\infty} X^2\frac{P(X)}{1-P(\emptyset)} = \frac{\overline{\overline{P}}}{1-P(\emptyset)},$$

$$\overline{G} = \sum_{k=\emptyset}^{\infty} kP(\emptyset)(1-P(\emptyset))^k = \frac{1-P(\emptyset)}{P(\emptyset)}.$$

We can find an expression for $\overline{\overline{G}} - \overline{G}$, which is needed in order to evaluate (5), as follows.

$$\overline{\overline{G}}-\overline{G} = \sum_{k=\emptyset}^{\infty}(k^2-k)P(\emptyset)(1-P(\emptyset))^k = \sum_{k=\emptyset}^{\infty} k(k-1)P(\emptyset)(1-P(\emptyset))^k$$

Let

$$GG(z) = \sum_{k=\emptyset}^{\infty} P(\emptyset)(1-P(\emptyset))^k z^k = \frac{P(\emptyset)}{1-z(1-P(\emptyset))},$$

by taking the second derivative of GG with respect to z, we get,

$$GG''(z) = \sum_{k=\emptyset}^{\infty} k(k-1)P(\emptyset)(1-P(\emptyset))^k z^{k-2}$$

$$= D_z^2\, P(\emptyset)[1-z(1-P(\emptyset))]^{-1}$$

$$= D_z\, P(\emptyset)(1-P(\emptyset))[1-z(1-P(\emptyset))]^{-2}$$

$$= \frac{2P(\emptyset)(1-P(\emptyset))^2}{[1-z(1-P(\emptyset))]^3}.$$

Then $\bar{\bar{G}}-\bar{G} = GG''(1)$

$$= \frac{2P(\emptyset)(1-P(\emptyset))^2}{[1-(1-P(\emptyset))]^3}$$

$$= \frac{2(1-P(\emptyset))^2}{P(\emptyset)^2}.$$

Substituting all these terms into (5), we obtain,

$$\sigma^2_{N_T} = \frac{\bar{P}^2}{(1-P(\emptyset))^2}\left(\frac{2(1-P(\emptyset))^2}{P(\emptyset)^2} - \frac{(1-P(\emptyset))^2}{P(\emptyset)^2}\right) + \frac{\bar{\bar{P}}}{1-P(\emptyset)}\,\frac{1-P(\emptyset)}{P(\emptyset)}.$$

$$= \frac{\bar{P}^2}{P(\emptyset)^2} + \frac{\bar{\bar{P}}}{P(\emptyset)}.$$

The expression (4) for $\sigma^2_{N_S}$ can be derived in a similar manner. QED

For example consider the class of (P_r, Q_s)-trees where $P_r(k) = 1/r+1$ for $\emptyset \le k \le r$, and $Q_s(c) = 1/s$ for $1 \le c \le s$.

$$E(N_T) = 1 + (r/2 - 1)/(1/r+1)\ ) = O(r^2),$$

$$\sigma^2_{N_T} = ((r/2)/(1/r+1) - 1)^2 + (r(2r+1)/6 - 1)/(1/r+1)$$

$$= O(r^4).$$

If (P_r, Q_s) were a good model of the trees generated by a particular branch and bound algorithm under a best-bound-first search strategy then the algorithm would have an expected

search tree size of $O(r^2)$ and a variance of $O(r^4)$. Note that the space complexity of best-bound search is the same order of magnitude as its time complexity.

Some comments are in order here. First, the time and space complexity of a best-bound-first search do not depend on Q, the distribution of arc costs. Only the relative ordering of nodes in a tree is important to a best-bound-first search, not the particular costs. This fact together with the assumption of mutual independence of the nodes of a (P,Q)-tree account for the absence of Q in (1) and (2).

Second, although the assumption of our model doesn't generally hold for interesting combinatorial minimization problems, (1) can be used to obtain an upper bound on the expected time complexity of a problem. Simply stated, the product of an upper bound on $\overline{P}$ (average degree of a random node) and an upper bound on $1/P(\emptyset)$ ($P(\emptyset)$ is the probability that a random node supplies a feasible solution) yields an upper bound on the expected time complexity. In chapter 7, we derive bounds along these lines for a subtour-elimination algorithm for the traveling salesman problem. Following the discussion above, upper bounds on $\overline{P}$ and $1/P(\emptyset)$, namely $O(\ln(n))$ (order of the natural logarithm of n) and n/e respectively, are multiplied to obtain an estimated upper bound ($\tilde{=}n\ln(n)/e$) on the expected time complexity.

Chapter 6.

Depth-First Search Strategies

6.1 Expected Time Complexity

The choice of which node to explore next in a depth-first search 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. In an ordered-depth-first search (odf) the sons are explored in the order of increasing cost as realized by the heuristic function in equation 2-(2a). If the sons are explored in the order of generation, then the search is called a generation-order-depth-first search (godf) as realized by the heuristic function in equation 2-(2b). Let $ET_{odf}(b)$ denote the expected size of the search tree generated by BB on a random (P,Q)-tree using the ordered-depth-first search strategy and given an initial bound of b. Let $ET_{godf}(b)$ denote the corresponding expected value for a generation-order-depth-first search. Expressions for these functions can be formulated fairly naturally as recurrence relations. Suppose that BB is searching a tree with the structure shown in Figure 6.1 where each subtree $T_1, T_2, \ldots, T_j$ may be regarded as a random (P,Q)-tree. Let $b_\emptyset$ be a finite initial bound (the bound on the root) and let b_i denote the bound at the top of the subtree T_i for $1 \leq i \leq j$. Then the expected size of the search tree for a

random tree with this structure is

$$1 + ET(b_1) + ET(b_2) + \ldots + ET(b_j).$$

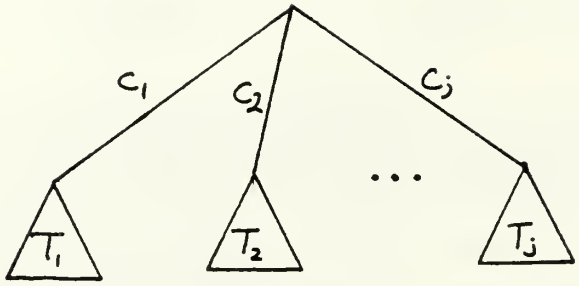
Each of the bounds b_i for $1 \leq i \leq j$ is less than $b_\emptyset$ indicating that a a recurrence relation may be set up. The next problem concerns the probability of a given bound occuring at a given node. Consider the tree in Figure 6.1. Given an initial bound of $b_\emptyset$, the bound on the subtree T_1 is $b_\emptyset - c_1$ so BB is expected to search $ET(b_\emptyset - c_1)$ nodes in T_1. $opt(T_1) = m$ with probability $\hat{O}(m)$ since it is a random (P,Q)-tree. The same holds for the other subtrees. Suppose that $opt(T_1) = m_1$. If $m_1 >= b_1$ then the search will not find the least cost leaf. On the other hand if $m_1 < b_1$ then the search will find the it. To summarize, the bound returned after searching the first subtree of the tree of Figure 6.1 with initial bound $b_\emptyset$ is $\min\{b_\emptyset, c_1+m_1\}$. The bound on the subtree T_2 is $\min\{b_\emptyset, c_1+m_1\}-c_2$. Continuing this reasoning one finds that the bound on the i^{th} subtree T_i is

$$b_i = \min\{b_\emptyset, c_1+m_1, c_2+m_2, \ldots, c_{i-1}+m_{i-1}\} - c_i \qquad (1)$$

where $m_1, m_2, \ldots, m_{i-1}$ denote the least cost leaves of the subtrees $T_1, T_2, \ldots, T_{i-1}$ respectively. ET evaluated with expression (1) yields the expected size of the subtree T_i.

The following function gives the expected size of T_i over all (P,Q)-trees. Let the functions $Wd_{odf}(b,i)$ and $Wd_{godf}(b,i)$ 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

Figure 6.1 A tree-top.



search and a generation-order-depth-first search respectively. An expression for $Wd_{godf}(b,i)$ can be found by essentially enumerating over all possible combinations of variables in (1).

$$Wd_{godf}(b,i) =$$

$$\sum_{c_1=1}^{\infty} \cdots \sum_{c_i=1}^{\infty} \sum_{m_1=\emptyset}^{\infty} \cdots \sum_{m_{i-1}=\emptyset}^{\infty} Q(c_1)\cdots Q(c_i)\hat{O}(m_1)\cdots\hat{O}(m_{i-1})*$$

$$ET_{godf}(\min\{b,c_1+m_1,\ldots,c_{i-1}+m_{i-1}\}-c_i) \qquad (2)$$

A tree with a structure as in Figure 6.1 will have expected size

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

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

$$ET_{godf}(b) = \sum_{j=\emptyset}^{\infty} P(j)\left(1+\sum_{i=1}^{j} Wd(b,i)\right)$$

$$= 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} Wd_{godf}(b,i). \qquad (3)$$

As stated, (2) is computationally intractable; however it can

refined to a more computable form as given in appendix A.

The order of examining the subtrees of figure 6.1 by an ordered-depth-first search is treated as follows. In an arbitrary (P,Q)-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 $c_1 \leq c_2 \leq \ldots \leq c_i$ 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 $c_j=c_{j+1}=\ldots=c_{j+k}$ then there is a repetition factor of k! due to this relation. In general

$$R_i\ (c_1,c_2,\ldots,c_i\ ) = i!/r_1!r_2!\ldots r_k!$$

where

$$c_1=\ldots=c_{r_1} < c_{r_1+1}=\ldots\ c_{r_1+r_2} <\ldots$$

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

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

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

$$Wd_{odf}(b,i) =$$

$$\sum_{c_1=1}^{\infty} \sum_{c_2=c_1}^{\infty} \ldots \sum_{c_{i-1}=c_i}^{\infty} Q(c_1)Q(c_2)\ldots Q(c_i) * R_i(c_1,c_2,\ldots,c_i)$$

$$\sum_{m_1=0}^{\infty} \cdots \sum_{m_{i-1}=0}^{\infty} \hat{O}(m_1)\cdots\hat{O}(m_{i-1})*$$

$$*ET_{odf}(\min\{b, c_1+m_1, c_2+m_2, \ldots, c_{i-1}+m_{i-1}\}-c_j) \qquad (4)$$

A tree with a structure as in Figure 6.1 will have expected size

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

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

$$ET_{odf}(b) = \sum_{j=0}^{\infty} P(j)(1+ \sum_{i=1}^{j} Wd_{odf}(b,i))$$

$$= 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} Wd_{odf}(b,i). \qquad (5)$$

Hereafter we will omit the subscript on ET and Wd except when necessary since all properties to be given have the same form for both. (i.e. the following theorems will hold with either subscript added). When the limit of the sequence ET(b) exists, it will be denoted ET(∞). Clearly when this limit exists we have the existence for each i of the limit of Wd(b,i), denoted Wd(∞,i). In corollary 4.1 we found necessary conditions for the existence of ET(∞), i.e. the existence of $\bar{P}$ and $G_{\bar{h}}$. Henceforth we will restrict our discussion to classes of (P,Q)-trees for which ET(∞) exists. We suspect that those classes of (P,Q)-trees for which the limit of ET does not exist are not particularly interesting in that they cannot be good models of combinatorial minimization problems.

Several results about these limits can be shown. Theorem 6.1 asserts that the expected size of a search tree is the same as the expected size of the search tree of the first subtree of the root.

Theorem 6.1. $ET(\infty) = Wd(\infty,1)$.

Proof: $$\lim_{b\to\infty} Wd(b,1) = \lim_{b\to\infty} \sum_{c=1}^{\infty} Q(c)ET(b-c) \quad \text{by (2)},$$

$$= \sum_{c=1}^{\infty} Q(c) \lim_{b\to\infty} ET(b-c)$$

$$= ET(\infty) \sum_{c=1}^{\infty} Q(c)$$

$$= ET(\infty). \quad \text{QED}$$

Theorem 6.2 gives an expression for $ET(\infty)$ which is similar to that for a best-bound-first search given by eq. 5-1. The quantity W is the expected number of nodes in the search tree except those in the first subtree.

Theorem 6.2. $ET(\infty) = W/P(\emptyset)$ where $W = 1 + \sum_{j=2}^{\infty} P(j) \sum_{i=2}^{j} Wd(\infty,i)$.

Proof: $$ET(\infty) = 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} Wd(\infty,i)$$

$$= 1 + Wd(\infty,1) \sum_{j=1}^{\infty} P(j) + \sum_{j=2}^{\infty} \sum_{i=2}^{j} Wd(\infty,i)$$

$$= 1 + ET(\infty)(1-P(\emptyset)) + \sum_{j=2}^{\infty} P(j) \sum_{i=2}^{j} Wd(\infty,i)$$

by theorem 6.1 thus,

$$ET(\infty) = [1 + \sum_{j=2}^{\infty} P(j) \sum_{i=2}^{j} Wd(\infty, i)]/P(\emptyset) = W/P(\emptyset). \quad \text{QED}$$

We can reason from theorem 6.2 to a lower bound on $ET(\infty)$ which was established by different means in theorem 4.2.

Proposition 6.2. $ET(\infty) \geq 1 + \overline{P}/P(\emptyset)$.

Proof: $$W = 1 + \sum_{j=2}^{\infty} P(j) \sum_{i=2}^{j} Wd(\infty, i)$$

$$\geq 1 + \sum_{j=2}^{\infty} P(j) \sum_{i=2}^{j} 1$$

$$= 1 + \sum_{j=2}^{\infty} (j-1)P(j)$$

$$= 1 + \sum_{j=2}^{\infty} jP(j) - \sum_{j=2}^{\infty} P(j)$$

$$= 1 + \overline{P} - P(1) - (1 - P(\emptyset) - P(1))$$

$$= \overline{P} + P(\emptyset).$$

Thus $ET(\infty) = W/P(\emptyset) \geq (P(\emptyset)+\overline{P})/P(\emptyset) = 1 + \overline{P}/P(\emptyset)$. QED

6.2 Time complexity as a function of the depth of the first leaf found in a depth first search tree.

The depth at which the first leaf is found in a depth-first search has a strong effect on the performance of the search. Intuitively if this depth is deep then the procedure will spend much of its time examining nodes in that part of the tree before returning to shallower levels where the true 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 what follows it is shown that a depth first search tree has a structure which is essentially linear in the depth of the first found solution. Let X(h) be the expected number of nodes in the search tree except those in the first subtree given that the first solution is found at depth h. X(Ø) is defined to be 1. See figure 6.2. Let S(h) be the expected number of nodes searched in a random (P,Q)-tree given that the first solution occurs at depth h. From these definitions one finds

$$S(h) = 1 + h + X(1) + X(2) + \ldots + X(h)$$

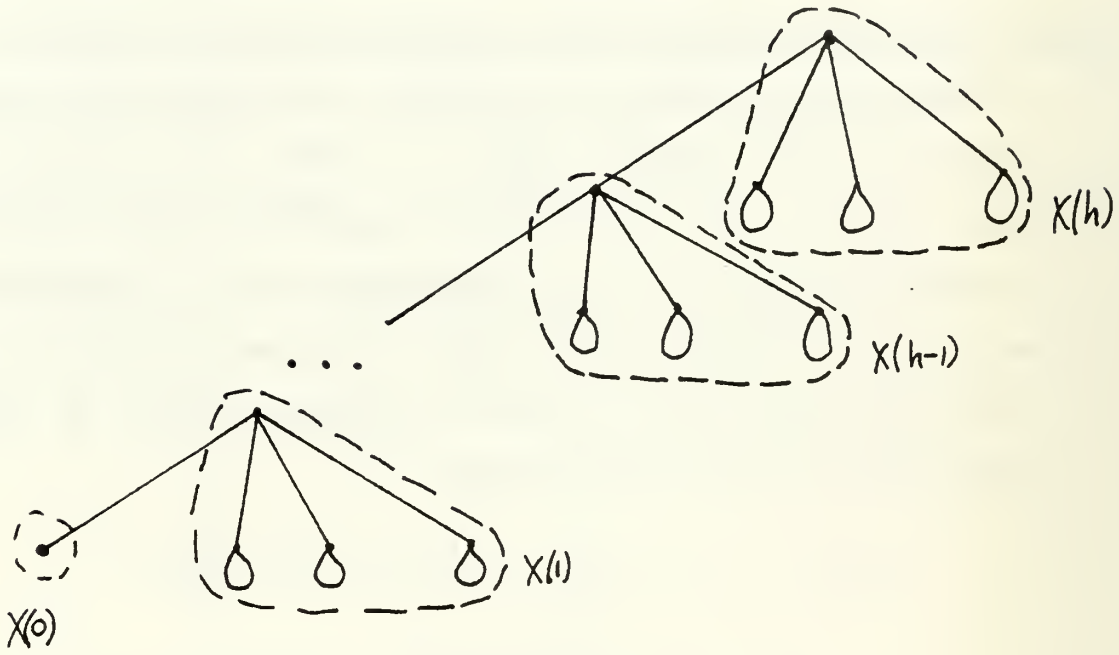
$$= 1 + h + \sum_{k=1}^{h} X(k)) \qquad (6)$$

In order to formulate an expression for X(d) an appropriate variant of $\hat{O}$ will be needed. Let $\hat{O}(d,i)$ denote the probability that opt(T)=i in a random (P,Q)-tree T given that the leftmost branch of T has length d. Similar reasoning to that which led to the expression 3-(4) for $\hat{O}(i)$ yields an expression for $\hat{O}(d,i)$. Again the method is to equate two expressions for the probability that opt(T)>i. One expression is

$$1 - \sum_{k=\emptyset}^{i} \hat{O}(d,k). \qquad (7)$$

Suppose now that the root has $j \geq 1$ sons with subtrees $T_1, T_2, \ldots, T_j$. This event occurs with probability $P(j)/1-P(\emptyset)$ for $d>\emptyset$ because the condition that the tree has a leftmost branch of length d disallows the possibility that the root has

Figure 6.2. The Structure of a Depth-First Search Tree.



zero sons. The probability that the root has j sons given that $j \neq \emptyset$ is $P(j)/1-P(\emptyset)$. By assumption subtree T_1 has a leftmost branch of length h-1 so $\hat{O}(h-1,i)$ applies to it, and the probability that $opt(root+T_1)>i$ is

$$1 - \sum_{s=1}^{i-1} \sum_{c=1}^{s} Q(c)\hat{O}(h-1,s-c)$$

For the subtrees $T_2,\ldots,T_m,\ldots,T_j$, the probability that $opt(T_m+root)>i$ is again given by 3-(3) since these may be random (P,Q)-trees. Thus summing over all trees T with leftmost branch of length h we have another expression for the probability that opt(T)>i.

$$\sum_{j=1}^{\infty} \frac{P(j)}{1-P(\emptyset)} \left[1- \sum_{s=1}^{i} \sum_{c=\emptyset}^{s-1} Q(s-c)\hat{O}(d-1,c)\right]\left[1- \sum_{s=1}^{i} \sum_{c=\emptyset}^{s-1} Q(s-c)\hat{O}(c)\right]^{j-1} \quad (8)$$

Expressions (7) and (8) may now be equated to establish a recurrence relation for $\hat{O}(d,i)$.

$$1 - \sum_{k=0}^{i} \hat{O}(d,k) =$$

$$\sum_{j=1}^{\infty} \frac{P(j)}{1-P(0)} \left[1 - \sum_{s=1}^{i} \sum_{c=0}^{s-1} Q(s-c)\hat{O}(d-1,c)\right]\left[1 - \sum_{s=1}^{i} \sum_{c=0}^{s-1} Q(s-c)\hat{O}(c)\right]^{j-1} \quad (9)$$

where $\hat{O}(0,0) = 1$, $\hat{O}(0,i) = 0$ for $i>0$.

The limit of this sequence of probability functions is expressed as follows.

$$1 - \sum_{k=0}^{i} \hat{O}(\infty,k) =$$

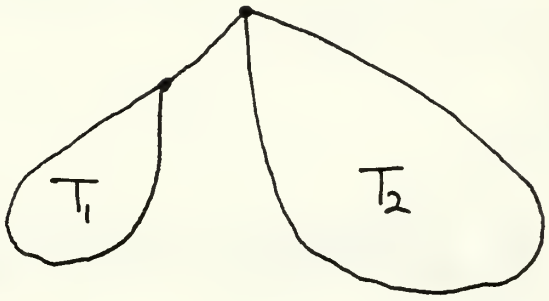
$$\sum_{j=1}^{\infty} \frac{P(j)}{1-P(0)} \left[1 - \sum_{s=1}^{i} \sum_{c=0}^{s-1} Q(s-c)\hat{O}(\infty,c)\right]\left[1 - \sum_{s=1}^{i} \sum_{c=0}^{s-1} Q(s-c)\hat{O}(c)\right]^{j-1} \quad (10)$$

Let LT_d denote the subset of (P,Q)-trees whose leftmost branchs have length d. Let RT denote the set of subtrees of nontrivial (P,Q)-trees formed by deleting the leftmost subtree of the root. An arbitrary $T \in LT_d$ can be realized as the grafting of a tree from LT_{d-1} (with attached arc) to a subtree from RT as in figure 6.3. An expression for X(d) can be found by summing over all combinations of this form. Let $Y_d(m)$ be the probability that a tree consisting of an arc plus a random tree from LT_d has a least cost leaf of cost m.

$$Y_d(m) = \sum_{k=1}^{m-1} \hat{O}(d,m)Q(m-k) \quad (11)$$

Let Z(m) be the expected size of the search tree of a random tree T in RT given an initial bound of m. Then X(d) has the

Figure 6.3. Formation of an arbitrary tree in LT_d.



$T_1 \epsilon LT_{d-1}$ $T_2 \epsilon RT$

form

$$X(d) = \sum_{k=1}^{\infty} Y_d(m)\ Z(m) \tag{12}$$

Proposition 6.3. For all $m \geq \emptyset$, $Z(m) \leq Z(m+1)$ and $\lim_{m\to\infty} Z(m)$ exists.

Proof: The first part of this lemma is just a particular case of theorem 2.2. The second part follows from our assumption that $ET(\infty)$ exists since for all m, $Z(m) \leq ET(m)$. QED

Let $Z(\infty)$ denote $\lim_{m\to\infty} Z(m)$.

Proposition 6.4. $\lim_{d\to\infty} X(d)$ is bounded above.

Proof: First note that $Y_\infty(m) = \sum_{k=1}^{m-1\wedge} O(\infty,k)Q(m-1)$,

thus $\lim_{d\to\infty} X(d) = \lim_{d\to\infty} \sum_{m=\emptyset}^{\infty} Y_d(m)Z(m)$

$$= \sum_{m=\emptyset}^{\infty} Y_\infty(m)Z(m)$$

$$\leq \sum_{m=\emptyset}^{\infty} Y_\infty(m)Z(\infty)$$

$$= Z(\infty) \sum_{m=\emptyset}^{\infty} Y_{\infty}(m) = Z(\infty). \quad \text{QED}$$

Let $X(\infty)$ denote $\lim_{d \to \infty} X(d)$. These propositions support one of our main results which states that $S(d)$ grows essentially linearly in d.

Theorem 6.3. $S(d)$ is bounded above and below by a linear function of d.

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

$$S(d) = 1 + \sum_{k=1}^{d} X(k) \quad \text{by (6)}$$

$$\geq 1 + \sum_{k=1}^{d} 1 = 1 + d.$$

Also,

$$S(d) = 1 + \sum_{k=1}^{d} X(k) \quad \text{by (6)}$$

$$\leq 1 + \sum_{k=1}^{d} Z(\infty) \quad \text{by proposition 6.4}$$

$$= 1 + Z(\infty)\, d.$$

Therefore, for all d, we have

$$1+d \leq S(d) \leq 1+Z(\infty)\, d. \quad \text{QED}$$

Theorem 6.3 can be interpreted as follows: The depth first search tree can be decomposed along the path from the root to the first found solution into groups of subtrees whose expected size is asymptotically constant (the i^{th} group consists of the $2^{nd}, 3^{rd}, \ldots, j^{th}$ subtrees below the i^{th} node on the path from the root to the first found solution). See Figure

6.2. Therefore the performance of branch and bound is expected to degrade linearly with the depth of the first found solution.

6.3. Expected Space Complexity of a Depth-First Search

Let ES(b) denote the expected space complexity of a random (P,Q)-tree when BB is given an initial bound of b. A recurrence relation for ES can be set up roughly analogous to the recurrence relation for ET. First we note that the space complexity must be at least 1 since initially the root is stored in memory. In terms of figure 6.1, suppose that the root has j sons. At this point in the search the root node has already been stored and removed from the queue so we have j nodes in memory. Let D(b,i) be the expected space complexity of the i^{th} subtree of the root when the initial value of the bound is b. When the root has j sons, the expected maximum amount of storage needed during the search of the i^{th} son is D(b,i) + j-i. The term j-i accounts for the number of nodes at depth 1 remaining in memory during the search of the i^{th} subtree. The maximum amount of memory used when the root has j sons is the maximum over D(b,i) + j-i for all $i \leq j$:

$$\max\{1,\ D(b,1)+j-1,\ D(b,2)+j-2,\ \ldots,\ D(b,j)\}.$$

This expression only has the value 1 when the root has zero sons since $D(b,1) \geq 1$ (when a subtree is searched, at least the root of the subtree was once in memory), thus we find,

$$ES(b) = \sum_{j=\emptyset}^{\infty} P(j)*\max\{1,D(b,1)+j-1,D(b,2)+j-2,\ldots,D(b,j)\}$$

$$= P(\emptyset)*1 + \sum_{j=1}^{\infty} P(j) \max_{i \leq j} \{D(b,i)+j-i\} \qquad (13)$$

where D(b,i) is the expected space complexity of the i^{th} subtree of the root given that the initial bound is b and can be formulated in the same way that we set up Wd(b,i):

$$D(b,i) = \sum_{c_1=1}^{\infty} \cdots \sum_{c_j=1}^{\infty} \sum_{m_1=\emptyset}^{\infty} \cdots \sum_{m_{j-1}=\emptyset}^{\infty} Q(c_1)\ldots Q(c_j)\hat{O}(m_1)\ldots\hat{O}(m_{j-1})*$$

$$ES(\min\{b, c_1+m_1, \ldots, c_{j-1}+m_{j-1}\}-c_j) \qquad (14)$$

This expression can be considerably simplified by noting that for the same reason that ET(b) is monotonically increasing in b (see theorem 2.2), so also is ES(b).

Proposition 6.3. For all $j \geq 1$, $D(b,1) \geq D(b,j)$.

Proof: $D(b,1) = \sum_{c_1=1}^{\infty} Q(c_1)ES(b-c_1)$

$$= \sum_{c_1=1}^{\infty} \cdots \sum_{c_j=1}^{\infty} \sum_{m_1=\emptyset}^{\infty} \cdots \sum_{m_{j-1}=\emptyset}^{\infty} Q(c_1)\ldots Q(c_j)\hat{O}(m_1)\ldots\hat{O}(m_{j-1})*ES(b-c_j)$$

$$\leq \sum_{c_1=1}^{\infty} \cdots \sum_{c_j=1}^{\infty} \sum_{m_1=\emptyset}^{\infty} \cdots \sum_{m_{j-1}=\emptyset}^{\infty} Q(c_1)\ldots Q(c_j)\hat{O}(m_1)\ldots\hat{O}(m_{j-1})*$$

$$ES(\min\{b, c_1+m_1, \ldots, c_{j-1}+m_{j-1}\}-c_j)$$

(since $b-c_j \geq \min\{b, c_1+m_1, \ldots, c_{j-1}+m_{j-1}\} - c_j$ and ES is monotonically increasing in b)

$$= D(b,j). \quad \text{QED}$$

As a result of the above lemma,

$$ES(b) = P(\emptyset) + \sum_{j=1}^{\infty} P(j) \max_{i \le j} \{D(b,i) + j-i\}$$

$$= P(\emptyset) + \sum_{j=1}^{\infty} P(j)(D(b,1) + j-1) \quad \text{by the lemma,}$$

$$= P(\emptyset) + \bar{P}-1 + (1-P(\emptyset))D(b,1)$$

$$= P(\emptyset) + \bar{P}-1 + (1-P(\emptyset)) \sum_{c=1}^{\infty} Q(c)\, ES(b-c). \qquad (15)$$

The limit of Depth can found as follows.

$$ES(\infty) = \lim_{b \to \infty} ES(b)$$

$$= \lim_{b \to \infty} (P(\emptyset) + \bar{P}-1 + (1-P(\emptyset)) \sum_{c=1}^{\infty} Q(c)ES(b-c))$$

$$= P(\emptyset) + \bar{P}-1 + (1-P(\emptyset)) \sum_{c=1}^{\infty} Q(c)ES(\infty)$$

$$= P(\emptyset) + \bar{P}-1 + (1-P(\emptyset))ES(\infty).$$

Thus $ES(\infty) = (P(\emptyset) + \bar{P}-1)/P(\emptyset) = 1 + (\bar{P}-1)/P(\emptyset)$. (16)

Suppose we need to estimate the maximum depth of explored nodes in a depth-first search. A slight alteration of the above arguments accomplishes this goal. Let Depth(b) = maximum depth of an explored node in a depth-first search of a random (P,Q)-tree. A recurrence relation for Depth(b) can be formulated by a slight alteration of (13).

$$Depth(\emptyset) = 1,$$

$$Depth(b) = 1 + \sum_{j=1}^{\infty} P(j) * \max_{j \le i} \{D(b,i)\} \qquad (17)$$

where D(b,i) is defined above in eq. (13). Again using proposition 6.4, we can simplify (16) to

$$\text{Depth}(b) = 1 + \sum_{j=1}^{\infty} P(j) * D(b,1)$$

$$= 1 + \sum_{j=1}^{\infty} P(j) \sum_{c=1}^{\infty} Q(c)\,\text{Depth}(b-c)$$

and

$$\lim_{b\to\infty} \text{Depth}(b) = \lim_{b\to\infty} 1 + \sum_{j=1}^{\infty} P(j) \sum_{c=1}^{\infty} Q(c)\,\text{Depth}(b-c)$$

$$= 1 + \sum_{j=1}^{\infty} P(j) * \text{Depth}(\infty)$$

$$= 1 + \text{Depth}(\infty) * (1 - P(\emptyset)).$$

Therefore

$$\text{Depth}(\infty)\,(1 - (1 - P(\emptyset))) = 1$$

which yields

$$\text{Depth}(\infty) = 1/P(\emptyset). \qquad (18)$$

Chapter 7.

An Application to the Traveling Salesman Problem

In this chapter we will show how the results of the previous chapters can be applied to a branch and bound algorithm called a subtour-elimination algorithm for solving asymmetric traveling salesman problems . The TSP may be stated as follows using the terminology of chapter 1: we want to find a constructive proof of $\exists x \forall y\ f(x) \leq f(y)$ where x,y are cyclic permutations of n objects or hamiltonian cycles on a complete directed graph with n nodes, and

$$f(x) = \sum_{i=1}^{n} c_{i,x_i}$$

where $[c_{i,j}]$ is an nxn asymmetric matrix giving the cost of the directed arc from i to j. The size of the instance is n.

A model of a particular branch and bound algorithm is an appropriate choice of P and Q functions parameterized by the problem size. We will develop such P and Q functions for the subtour-elimination algorithm described in chapter 2 by studying the behavior of the algorithm on the initial feasible space of permutations. Again, this algorithm makes use of a relaxation of the requirement that feasible objects be cyclic permutations, and the initial feasible set is the set S_n of permutations of n objects. The set S_n is a symmetric set in the sense

that for any given pair π_1, $\pi_2 \in S_n$ there is a relabelling (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.

We have 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 matrix. 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

$$P(0) = (n-1)!/n! = 1/n. \qquad (1)$$

Let

$$H_n = \sum_{k=1}^{n} 1/k \qquad \text{for all } n.$$

The numbers H_n are called harmonic numbers [Knuth 1969] and occur frequently in the analysis of algorithms. There is a well-known asymptotic expansion of these numbers

$$H_n = \ln(n) + \gamma + 1/(2n) - O(n^{-2}) \qquad (2)$$

where $\gamma = 0.577...$ is called eulers constant. From (2), we obtain the following bounds on H_n,

$$\ln(n) + \gamma < H_n < \ln(n) + \gamma + 1/(2n).$$

The following theorem helps us obtain asymptotic values for

P(k) when $k \geq 1$.

Theorem 7.1. 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

$$\lim_{n\to\infty} S(n,k) = e^{-H_k} \quad \text{for } k \geq 1.$$

Proof: We will 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, $S(n,0) = 1 = e^{-H_0}$.

Assume now that

$$\lim_{n\to\infty} S(n,k-1) = e^{-H_{k-1}} \quad \text{for some } k>0.$$

The probability S(n,k) can be formulated as (1/n!)*(number of permutations whose subcycles all have order greater than k). We will use the principle of inclusion-exclusion [Liu 1968] in order to get S(n,k) essentially by subtracting 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 tnat 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 $\binom{n}{k}\binom{n-k}{k}$ $\ldots\binom{n-mk+k}{k}$ ways to pick m such sets, $(k-1)!^m/m!$ ways to arrange these sets into 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 we find

$$S(n,k) = \frac{1}{n!}\sum_{m=0}^{n/k}(-1)^m\frac{(k-1)!^m}{m!}\binom{n}{k}\binom{n-k}{k}\ldots\binom{n-mk+k}{k}(n-mk)!S(n-mk,k-1)$$

$$= \sum_{m=0}^{n/k}\frac{(-1)^m(k-1)^m}{n!}\frac{n!}{m!}\frac{(n-k)!}{k!(n-k)!}\frac{}{k!(n-2k)!}\cdots\frac{(n-mk+k)!}{k!(n-mk)!}(n-mk)!S(n-mk,k-1)$$

$$= \sum_{m=0}^{n/k}\frac{(-1/k)^m}{m!}S(n-mk,k-1).$$

When we take the limit of this function, we get

$$\lim_{n\to\infty} S(n,k) = \lim_{n\to\infty} \sum_{m=0}^{n/k} \frac{(-1/k)^m}{m!} S(n-mk,k-1).$$

$$= \sum_{m=0}^{\infty} \frac{(-1/k)^m}{m!} \lim_{n\to\infty} S(n-mk,k-1).$$

$$= \sum_{m=0}^{\infty} \frac{(-1/k)^m}{m!} e^{-H_{k-1}} \quad \text{(by induction hypothesis)}$$

$$= e^{-H_{k-1}} e^{-1/k}$$

$$= e^{-H_k}. \qquad \text{QED}$$

An immediate corollary of theorem 7.1 is the well-known result that there are $n!S(n,1)$ which is asymptotic to $n!e^{-H_1} = n!/e$ n-permutations which do not have any 1-cycles (this is known as the problem of derrangements [Riordan 1958; Liu 1968]). Our intended application of theorem 7.1 is the probability that the least cost permutation has k sons (its smallest subcycle is of order k).

Theorem 7.2. The asymptotic probability that the least cost n-permutation on a random cost matrix has a smallest order cycle of order k is

$$\lim_{n\to\infty} P_n(k) = e^{-H_{k-1}} - e^{-H_k}. \tag{3}$$

Proof: We have already noted that each n-permutation is equally likely to be the least cost permutation over a random cost matrix. 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 7.1.

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

$$P'(0) = (n-1)!/(n!/e) = e/n. \qquad (4)$$

It cannot be shown that (4) is asymptotically correct as easily as (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 supports (4). See Table 1 and Table 2. Next we might ask how P(k) is affected by this alteration of the cost matrices. Let

$$P'(k) = Pr(\text{the smallest cycle of } \pi \text{ has order } k \mid k>1)$$

$$= P(k)/(1-P(1))$$

$$= (e^{-H_{k-1}} - e^{-H_k})/(1 - (1-1/e))$$

$$= e(e^{-H_{k-1}} - e^{-H_k}). \qquad (5)$$

Given (5), we can find an upper bound on $\bar{P}'$.

$$\bar{P}' = \sum_{k=2}^{n/2} kP'(k)$$

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

$$= e[2(e^{-H_1}-e^{-H_2})+3(e^{-H_2}-e^{-H_3})+\ldots +(n/2)(e^{-H_{n/2-1}}-e^{-H_{n/2}})]$$

$$= e[e^{-H_1} - (n/2)e^{-H_{n/2}} + \sum_{k=1}^{n/2-1} e^{-H_k}]$$

$$< e[e^{-1} - (n/2)(2/n)e^{-\gamma_e-1/2(n/2)} + \sum_{k=1}^{(n/2)-1} e^{-H_k}]$$

(we have made use of the bounds obtained above on H_k),

$$= 1 - e^{1-\gamma_e-1/n} + e^{1-\gamma}H_{(n/2)-1}. \qquad (6)$$

With $P'(0)$ and $\bar{P}'$ we can now test our estimated expected time for solving randomly generated traveling salesman problems using a subtour-elimination algorithm under the best-bound-first search strategy. Inserting the bounds (5) and (6) into equation 3-9, we have

$$E(N_T) = 1 + \bar{P}/P(0)$$

$$< 1 + [1 - e^{1-\gamma_e-1/n} + e^{1-\gamma}H_{(n/2)-1}]/(e/n)$$

$$= 1 + n/e - ne^{-\gamma_e-1/n} + e^{-\gamma}nH_{n/2-1} \qquad (7)$$

$$= O(n\ln(n)) \qquad (7')$$

For each of the nodes counted by N_T, we solve the corresponding assignment problem. As stated above in chapter 2, these assignment problems take $O(n^3)$ time at the root (the initial problem) and $O(n^2)$ time for subsequent problems. The only other factor in the running time of the algorithm is the time required to maintain the priority queue. Using available techniques for implementing priority queues [Aho, Hopcroft, and Ullman 1974] the time required to insert or access a node in the queue when n nodes are in it is $O(\ln(n))$. The access and insertion time per node for any branch and bound algorithm depends on the order of magnitude of the space complexity, the maximum number of nodes in storage during the search. For the subtour-elimination algorithm the space complexity is $O(n\ln(n))$, thus the mean queue maintenence time is

$$O(\ln(n\ln(n))) = O(\ln(n) + \ln\ln(n)) = O(\ln(n)).$$

Putting these quantities together , we expect the running time of the subtour-elimination algorithm to be

$$1*O(n^3) + O(n\ln(n))*O(n^2)*O(\ln(n)) = O(n^3\ln^2(n)).$$

In table 1, the bounds (7) are computed for several values of n. Compared with these values are empirical values of $E(N_T)$ found by averaging N_T from 1000 randomly generated traveling salesman problems 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

TABLE 1. Data from the solution of randomly generated traveling salesman problems by a subtour-elimination algorithm using a best-bound-first search strategy compared with theoretical estimates of the corresponding values.

Size of problem	No. of problems solved	Sample mean search tree search	Mean search size by eq. 7-7 tree	Sample $\bar{P}$ at root	Bound on $\bar{P}$ by eq.7-6	Sample P(0) at root	P(0) by eq.7-4 (= e/n)
10	1000	6.48	11.29	2.03	2.80	.261	.272
15	1000	12.28	19.27	2.58	3.31	.186	.181
20	1000	19.63	29.44	3.09	3.87	.153	.136
25	790	31.19	39.10	3.50	4.14	.106	.109

and uniformly distributed random integers between 1 and 1000 in each entry. The diagonal entries were set to a very large number. Table 2 presents data on the probabilities of the various branching factors of nodes at different depths in the search tree. Notice that P(0) seems to increase monotonically with depth. This provides evidence that e/n is indeed a lower bound on P(0). (for n=20, we have e/20 = 0.136... compared with 0.135 for P(0) at depth 0). The most dramatic changes take place between depth 0 and depth 1. In particular P(0) almost doubles and P(2) roughly halves. Note that at depth 0 the sample mean is 3.018 whereas our estimated mean using (5) is,

$$\sum_{k=2}^{10} ke(e^{-H_{k-1}} - e^{-H_k}) = 2.982\ldots .$$

It was first suggested by Bellmore and Malone [Bellmore and Malone 1971] that subtour-elimination algorithms exhibit polynomial expected time behavior on randomly generated problems. The proof of this behavior entirely rests on showing

TABLE 2. Data from the solution of 790 randomly generated assymetric traveling salesmen 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.

	Sample values of P at depth m in the search tree										
	m= 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(1)	0.386	0.195	0.141	0.122	0.115	0.101	0.090	0.090	0.084	0.090	0.097
P(2)	0.180	0.159	0.133	0.123	0.115	0.106	0.105	0.098	0.099	0.101	0.088
P(3)	0.105	0.103	0.109	0.102	0.097	0.096	0.090	0.089	0.087	0.081	0.089
P(4)	0.063	0.085	0.083	0.082	0.081	0.084	0.087	0.091	0.098	0.096	0.096
P(5)	0.037	0.055	0.065	0.068	0.071	0.072	0.074	0.066	0.067	0.067	0.070
P(6)	0.032	0.049	0.049	0.056	0.056	0.060	0.061	0.054	0.065	0.051	0.047
P(7)	0.030	0.050	0.050	0.052	0.053	0.055	0.057	0.062	0.055	0.055	0.056
P(8)	0.020	0.042	0.046	0.049	0.053	0.052	0.051	0.050	0.055	0.054	0.060
P(9)	0.009	0.026	0.023	0.023	0.023	0.026	0.027	0.024	0.025	0.033	0.031
$\bar{P}$	3.015	3.435	3.324	3.338	3.367	3.314	3.373	3.353	3.345	3.302	3.341

that (1) or (4) is a lower bound on P(0) (the probability that a randomly picked node in a randomly picked search tree has zero sons) or, as noted in [Rinnooy Kan and Lenstra 1978], that $O(n^{-c})$ for any constant c is a lower bound on P(0). This important result is the object of current research.

For some time now it has been taken as a general guide that if an algorithm runs in polynomial time then it is a tractable problem. If the algorithm runs in superpolynomial time then it is intractable. So far no polynomial time algorithm has been found for any NP-complete problem, so they are considered intractable. But the NP-complete (and NP-hard) problems are intractable only in terms of a worst-case bound; no known algorithm is guaranteed to halt with a solution within a polynomial amount of time at present. Here though, we have in the traveling salesman problem an NP-complete problem which seems to be solveable on the average in polynomial time. Thus many instances of the traveling salesman problem can be tractably solved but a few hard instances of the problem cause intractable behavior. The existence of such problems takes some of the sting from the possibility that P≠NP. One might reasonably ask whether all NP-complete problems are solveable in polynomial expected time. In fact we might define a new class of problems called EP which are solveable in polynomial time on the average. Certainly P EP and it seems that the traveling salesman problem is in EP. Goldberg [Goldberg 1979] has recently shown that the satisfiability problem seems to be solveable in polynomial expected time. A proof that a problem $\pi \in NP$

is not in EP constitutes a proof that P≠NP since P is a subset of EP. The problem of course with this definition of EP is the need to define an appropriate probability measure on a problem. Pathological probability measures can be found which emphasize the hardest instances of a problem (thus making the problem seem hard), or emphasize the easiest instances (making the problem seem easily solveable). Goldberg chose the reasonable course of showing that the satisfiablity problem is solveable in polynomial expected time under several different probability measures on the problem. So a meaningful definition of EP awaits further insight into what we mean by a natural or reasonable probability measure on a problem.

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

$$Q_n(k) = (0.00054n)\left(\frac{1}{1+0.00054n}\right)^k \qquad (8)$$

where n is the size of the class of problems. Table 3 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 chapter 6. 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 (8) for Q and our formulas (4)

and (5) for P in computing ET, in the column marked ET(1000). Note that ET(1000) using this P function underestimates the sample mean. According to our investigation of the best-bound-first search strategy, we need an upper bound on $\overline{P}$ and a lower bound on P(0) in order to get an estimate which bounds our sample mean from above. While $\overline{P}$ computed from (4) and (5) is a good estimate for the mean branching factor over all root nodes, it does not seem to be a good enough bound on the average branching factor at other depths according to Table 2. We obtain good upper bounds by amending P as follows: Halve P(2) and distribute the difference over P(3), P(4), ..., P($\lfloor n/2 \rfloor$). We retain P(0) = e/n. In this way the mean of $\overline{P}$ has been increased and the lower bound on P(0) remains. The bounds obtained using this P function in ET are given in the column labeled ET'(1000) in Table 3.

Theorem 6.3 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 leftmost path in the search tree. At the same time that we found the sample mean search tree size of random traveling salesman problems above, we sampled the search tree size as a function of the length of the leftmost branch of the tree. This data is presented in Table 4 and graphically in Figure 7.1. The data in Figure 7.3 clearly shows the linear growth of the mean search tree size for as far as the sample means are meaningful.

TABLE 3. Data from the solution of randomly generated traveling salesman problems by a subtour-elimination algorithm using a depth-first-search strategy and given an initial bound of 1000 (1000 above and beyond the lower bound on the root). This data is compared with estimates computed from our model.

Size of problem	No. of problems solved	Sample mean search tree size	ET bound =1000	ET' bound =1000	Sample mean stack depth	Stack depth bound eq. 6-18
10	1000	10.36	11.06	13.45	2.98	3.68
15	1000	35.82	30.03	38.61	4.83	5.52
20	790	81.85	64.40	88.72	5.50	7.36

TABLE 4. Data from randomly generated traveling salesman problems giving the mean time complexity as a function of the length of the leftmost path in the search tree.

Size of problem	No. of problems solved	Mean search tree size when the leftmost branch has length k										
		k=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	780	1	17	35	85	94	128	160	182	207	299	...

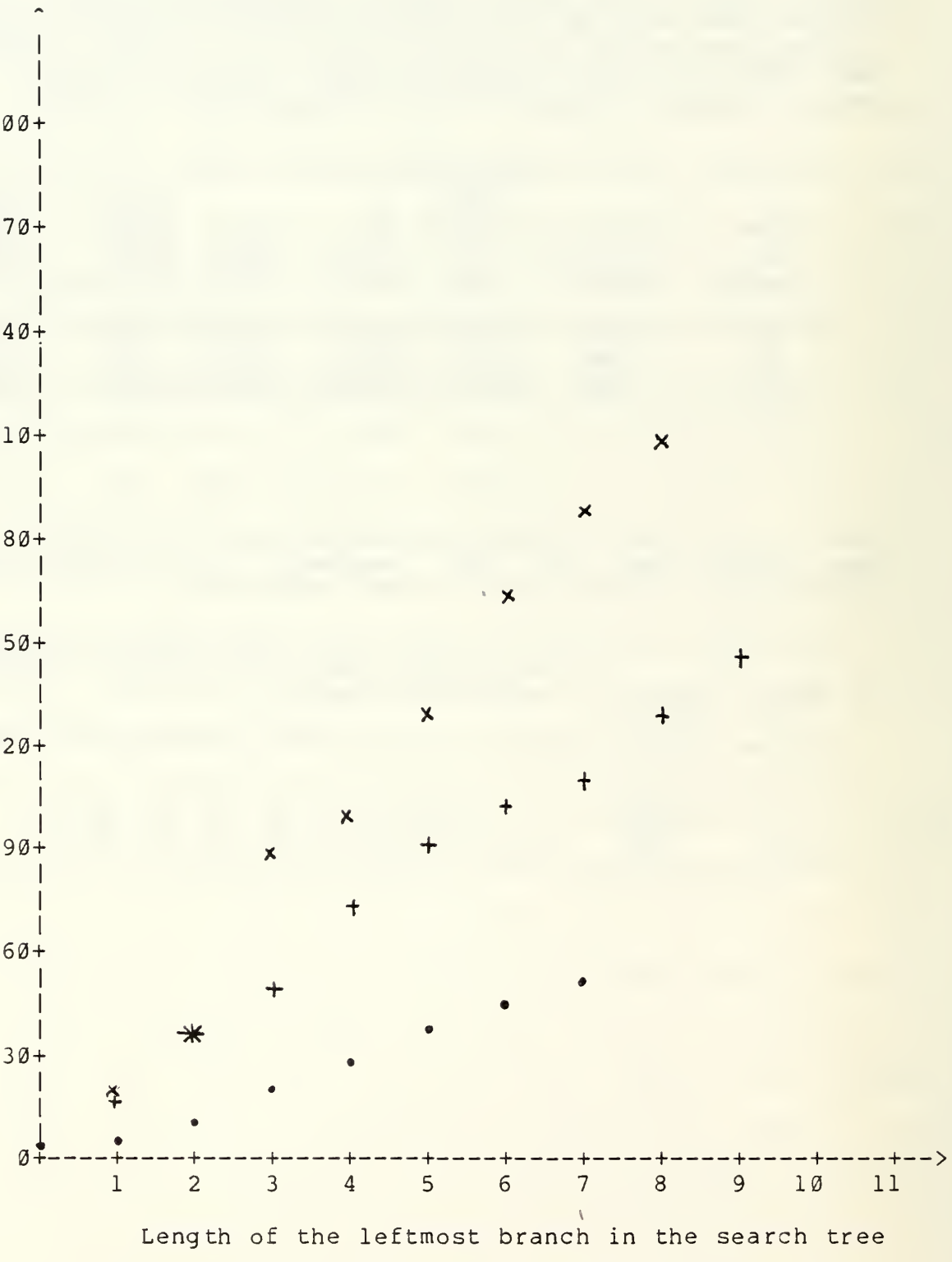


Figure 7.1. The data from Table 4 plotted, showing the growth of the sample mean search tree size as a function of the length of the leftmost branch in the search tree. The circles, pluses, and x's represent data points from traveling salesman problems of size 10, 15, and 20 respectively. The problems were solved by a subtour-elimination algorithm using a depth-first search strategy.

Conclusions

Chapter 8

We have studied a model of branch and bound algorithms and derived expressions for the mean space and time requirements for several search strategies. It has been shown that in our model the best-bound-first search strategy is optimal in terms of our measures of time and space complexity. The results we have obtained are essentially order of magnitude results and it may turn out in practice that the constants associated with the order of magnitude for a given algorithm make a difference as far as the choice of search strategy. In a best-bound-first search a unit of storage may be quite large if, for example, we need to store an entire matrix as in an integer linear program, since enough information must be stored in order to restart the search from each unexplored node. On the other hand, in a depth-first search, the context of the search is stored in the ancestors of a node, so comparatively little information need be stored per node. For this reason the best-bound-first search strategy, although widely recognized as optimal in terms of time complexity, is viewed as excessively space-consuming. Another complaint against the best-bound-first search strategy is the inefficiency caused by the bookkeeping involved. But there are efficient data struc-

tures and associated routines for their manipulation available now which can be used to implement this strategy; one, mentioned in chapter 2, being the priority queue [Aho, Hopcroft, and Ullman 1974]. Of the roughly 45 minutes of CPU on an IBM 370/165 spent in producing the data of table 1, less than 6 seconds were spent maintaining the priority queue. Breadth-first is not usually a practical choice of search strategy for branch and bound algorithms because it has the disadvantages of best-bound-first and depth-first without their advantages. Breadth-first is like best-bound-first in that all nodes in memory are effectively the roots of different search trees and for each node all information neccesary for starting up the associated subproblem must be stored. This means that breadth-first search has a large constant associated with its space complexity. On the other hand it is like a depth-first search in that it is easy to construct a tree for which a best-bound search explores less nodes than breadth-first search. So it is nonoptimal in terms of time complexity. A breadth-first search is reasonable however when it is known or suspected that the optimal solution is found at a shallow depth.

Our model is particularly suited for modelling relaxation procedures, where there is some chance that any node in the search tree of a random problem from a class may produce a feasible solution. The success of the assignment problem relaxation for solving assymetric 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 hard combinatorial problems might begin by looking for suitable relaxations and fast algorithms for solving them. The search for fast approximate algorithms for hard combinatorial problems can also benefit from the use of relaxations of a problem. A relaxed solution to a problem may have many of the components of an optimal feasible solution. A heuristic restructuring of the relaxed solution might produce a feasible solution of near optimal cost.

APPENDIX

Several of the results of this thesis have been formulated as somewhat complex recurrence relations. In this section we show how two of these recurrence relations can be broken down into simpler relations which aid in the computation of their sequences.

In chapter 3 the function $\hat{O}$ was introduced in the form

$$1 - \sum_{k=\emptyset}^{i} \hat{O}(k) = \sum_{j=1}^{\infty} P(j) * [1 - \sum_{s=1}^{i} \sum_{c=1}^{s} Q(c)\hat{O}(s-c)]^{j} \qquad (1)$$

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

Let

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

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

$$= 1 - \sum_{s=1}^{i} E(s) = G(i-1) - E(i),$$

$$B(i) = \sum_{j=1}^{\infty} P(j)G(i)^{j},$$

$$\hat{O}(i) = B(i-1) - B(i).$$

Note that $B(i) = 1 - \sum_{k=\emptyset}^{i} \hat{O}(k)$, therefore $B(i-1) - B(i) = \hat{O}(i)$. In terms of these functions the computation of $\hat{O}$ proceeds as given in the high level algorithm of figure A.1. For some Q functions,

Figure A.1. An algorithm for computing $\hat{O}$ on the range $0,1,2,\ldots,limit$, given the probability functions P and Q.

```
begin
B(0) := 1-P(0);
Ô(0) := P(0);
for i:=1 until limit;
    begin
        E(i) := Σ_{c=1}^{i} Q(c)Ô(i-c);
        G(i) := G(i-1) - E(i);
        B(i) := Σ_{j=1}^{∞} P(j)G(i)^j;
        Ô(i) := B(i-1) - B(i);
    end
end
```

E(i) may be easily expressed as a recurrence relation, further simplifying the computation of $\hat{O}$ (and the computation of ET given below). For example if Q is geometric, $Q(c) = rs^c$, then,

$$E(i) = \sum_{c=1}^{i} rs^c * \hat{O}(i-c) = 1/s \sum_{c=1}^{i} rs^c * \hat{O}((i+1) - (c+1))$$

$$= (1/s) \sum_{c=2}^{i+1} rs^c * \hat{O}(i+1-c) = (1/s)(E(i+1) - rs\hat{O}(i))$$

therefore,

$$E(i+1) = sE(i) + rs\hat{O}(i).$$

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

$$ET(b) = 1 + \sum_{j=1}^{\infty} P(j) \sum_{i=1}^{j} Wd(b,i) \qquad (2)$$

where Wd(b,i) =

$$\sum_{c_1=1}^{\infty} \cdots \sum_{c_i=1}^{\infty} \sum_{m_1=\emptyset}^{\infty} \cdots \sum_{m_{i-1}=\emptyset}^{\infty} Q(c_1)\ldots Q(c_i)\hat{O}(m_1)\ldots\hat{O}(m_{i-1})*$$

$$*ET(\min\{b, c_1+m_1, c_2+m_2, \ldots, c_{i-1}+l_{i-1}\} - c_i) \qquad (3)$$

Essentially, Wd(b,i) has the form

$$Wd(b,i) = \sum_{k=1}^{\infty} R(b,i,k) \sum_{c_i=1}^{\infty} Q(c_i)ET(k-c_i) \qquad (4)$$

where R(b,i,k) = probability that $k = \min\{b, c_1+l_1, \ldots, c_{i-1}+l_{i-1}\}$. (the term c_j+l_j is the cost of the least cost leaf in the j^{th} subtree below the root; c.f. Figure 3b). In other words, k is the value of the bound immediately after the $i-1^{st}$ subtree has been explored. R(b,i,k) may be formulated easily as follows: We have 2 cases, either k=b or k<b. The probability that k=b is

$$R(b,i,b) = Pr(c_1+l_1 \geq b)*Pr(c_2+l_2 \geq b)*\ldots*Pr(c_{i-1}+l_{i-1} \geq b)$$

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

$$R(b,i,b) = G(b)^{i-1}. \tag{5}$$

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,

$$R(b,i,m) = (i-1)*E(m)*G(m)^{i-2} \tag{6}$$

Substituting (5) and (6) into (4) we get

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

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

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

$$= H(b-1) + (i-1)E(b-1)G(b-1)^{i-2}D(b-1) \tag{7}$$

we have

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

Looking again at (2), we see that we need partial sums of Wd(b,i), so let

$$W(b,i) = \sum_{j=1}^{i} Wd(b,i) = W(b,i-1) + Wd(b,i)$$

$$= W(b,i-1) + H(b,i) + G(b)_{i-1}D(b) \qquad (8)$$

Putting all these pieces together, we can compute ET as in Figure A.2. The infinities which appear in the algorithms of figures A.1 and A.2 only come into play when P has an infinite range, i.e., 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 2-dimensional arrays W and H can be reduced to 1-dimensional arrays.

Figure A.2. An algorithm for computing ET(b) for the expected size of a depth-first search tree given P, Q, and $\hat{O}$.

```
begin
ET(Ø) := 1;
for all b, W(b,Ø) := Ø;
for all b, H(b,Ø) := Ø;
G(Ø) := 1;
E(Ø) := Ø;

for b := 1 until limit
    begin
        for i:=1,...,∞
            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);

                  b
        E(b) :=   Σ  Q(c)Ô(b-c);
                 c=1
                  b
        D(b) :=   Σ  Q(c)ET(b-c);
                 c=1

        for i:=1 until ∞;
            W(b,i) := W(b,i-1) + H(b,i) + G(b)^(i-1)D(b);
                      ∞
        ET(b) := 1 +  Σ  P(j)W(b,j);
                     j=1
    end
end
```

References

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1974), The Design and Analysis of Algorithms. Addison-Wesley, Reading, MA, 1974.

Beardwood, J. Halton, J.H., and Hammersley, J.M.(1959), The Shortest Path Problem Through Many Points. Proc. Camb. Phil. Soc., 55(1959), 299-327.

Bellman, R.E. (1957), Dynamic Programming. Princeton University Press, Princeton, NJ, 1957.

Bellmore M., and Malone, J.C.(1971), Pathology of Traveling Salesman Subtour-Elimination Algorithms. Operations Research 19, (1971), 278-307.

Bellmore, M., and Nemhauser, G.L.(1968), The Traveling Salesman Problem: A Survey. Operations Research 16, (1968), 538-558.

Eastman, W.L. (1958), Linear Programming with Pattern Constraints. Unpublished Ph.D. Dissertation, Harvard Univ., Cambridge, MA, 1958.

Feller, W. (1950), An Introduction to Probability and its Applications, Vol. I. John Wiley & Sons Inc., New York, NY, 1950.

Fillmore, J.P., and Williamson, S.C. (1974), On Backtracking: A Combinatorial Description of the Algorithm. SIAM J. of Computing, Vol. 3, No. 1, March 1974, 41-55.

Fox, B.L., Lenstra, J.K., Rinnooy Kan, A.H.G., and Schrage, L.E. (1978), Branching from the largest Upper Bound. European Journal of Operational Research 2, (1978), 191-194.

Garfinkel, R.S. (1973), On Partitioning the Feasible Set in a Branch and Bound Algorithm for the Assymetric Traveling Salesman Problem. Operations Research 21(1973), 340-343.

Garfinkel, R.S., and Nemhauser, G.L. (1972), Integer Programming. John Wiley, New York, NY, 1972.

Goldberg, A. (1979), Average Case Complexity of the Satisfiability Problem. Proceedings of the Fourth Workshop on Automated Deduction, Austin, Texas, 1979, 1-6.

Golomb, S.W., amd Baumert, L., (1965), Backtrack Programming. JACM 12(1965), 516-524.

Harris, T.E. (1963), The Theory of Branching Processes. Springer-Verlag, Berlin, 1963.

Hart, P., Nilsson, N.J., and Raphael, B., (1968), A Formal Basis for the Heuristic Determination of Minimal Cost Paths. IEEE Trans. System Sci. Cybernetics, ssc-4, 1968, 100-107.

Held, M., and Karp, R.M. (1971), The Traveling Salesman Problem and Minimum Spanning Trees: Part II. Mathematical Programming 1, pg 6-25, 1971.

Ibaraki, T. (1976), Theoretical Comparison of Search Strategies in Branch and Bound Algorithms. International Journal of Computer and Information Sciences 5, 4(1976), 315-344.

Ibaraki, T. (1977), The Power of Dominance Relations in Branch and Bound Algorithms. J. ACM 24, 2(1977), 264-279.

Ibaraki, T. (1978), Branch and Bound Procedure and State-Space Representation of Combinatorial Optimization Problems. Information and Control 36, (1978), 1-27.

IBM Mathematical Programming System Extended (MPSX) Mixed Integer Programming (MIP) Program Description (1971), Technical Publication SH20-0908-1, White Plains, NY, 1971.

Ignall, E. and Schrage, L. (1965), Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. Operations Research 11, pg 400-412, 1965.

Jordan, C. (1950), Calculus of Finite Differences. Chelsea Publishing Co., New York, NY, 1950.

Kanal, L.N. (1978), State Space Models for Pattern Recognition. Laboratory for Pattern Analysis, Univ. of Maryland, College Park, MD 20742, 1978.

Karp, R.E. (1972), Reducibility Among Combinatorial Problems. in Complexity of Computer Computations. R.E. Miller and T.W. Thatcher, eds., Plenum Press, New York, NY, 1972, 85-104.

Karp, R.M. (1976), The Probabilistic Analysis of Some Combinatorial Search Algorithms. in Proceedings of the Symposium on New Directions and Recent Results in Algorithms and Complexity. Academic Press, 1976.

Klee, V., and Minty, G.J. (1970), How Good is the Simplex Algorithm? Mathematical Note No. 643, Boeing Scientific Research Laboratories, 1970.

Knuth, D.E. (1968), Fundamental Algorithms: The Art of Computer

Programming 1, Addison-Wesley, Reading, MA, 1968.

Knuth, D.E. (1974), Estimating the Efficiency of Backtrack Algorithms. Mathematics of Computation, 29(1975), 121-136.

Knuth, D.E., and Moore, R.W. (1975), An Analysis of Alpha-Beta Pruning. Artificial Intelligence, 6(1975), 293-326.

Kohler, W.H., and Steiglitz, K. (1974), Characterizations and Theoretical Comparison of Branch and Bound Algorithms for Permutation Problems. J. ACM 21, 1(1974), 140-156.

Kohler, W.H., and Steiglitz, K. (1975), Enumerative and iterative Computational Approaches, in Computer and Job-Shop Scheduling Theory, E. Coffman, Ed., Chapter 6, Wiley Interscience, New York, NY, 1975.

Lawler, E.L., and Wood, D.E. (1966), Branch and Bound Methods: A Survey. Operations Research 14, 4(1966), 699-719.

Lawler, E. (1976), Combinatorial Optimization: Networks and Matroids. Holt, Rinehart, and Winston, New York, NY, 1976.

Lehmer, D. (1959), The Machine Tools of Combinatorial Mathematics. in Applied Combinatorial Mathematics. Ed. Beckenbach, John Wiley Co., 1964, 5-31.

Lenstra, J.K., and Rinnooy Kan, A.H.G. (1978), On the Expected Performance of Branch and Bound Algorithms. Operations Research 26, 2(1978), 347-349.

Lin, S. (1965), Computer Solutions of the Traveling Salesman Problem. Bell System Technical Journal, Vol. 44, 1965, 2245-2270.

Liu, C.L. (1968), Introduction to Combinatorial Mathematics. McGraw-Hill Book Co., New York, NY, 1968.

Luenberger,D. (1973), Introduction to Linear and Nonlinear Programming. Addison-Wesley Pub. Co., Reading, MA, 1973.

Mode, C.J. (1971), Multitype Branching Processes: Theory and Applications. American Elsevier Publishing Company Inc., New York, NY, 1971.

Mitten, L.G. (1970), Branch and Bound Methods: General Formulation and Properties. Operations Research 18, (1970), 24-34.

Morin T.L., and Marsten, R.E. (1976), Branch and Bound Strategies for Dynamic Programming. Operations Research 24, pg 611-629, 1976.

Morin T.L., and Marsten, R.E. (1978), A Hybrid Approach to Discrete Mathematical Programs. Mathematical Programming 14,

pg 21-40, 1978.

Munkres, J. (1957), Algorithms for the Assignment and Transportation Problems. J. Soc. Indust. Appl. Math., Vol 5, No. 1, March, 1957, 32-38.

Nilsson, N.J. (1971), Problem Solving Methods in Artificial Intelligence. McGraw-Hill, NY, 1971.

Papadimitiou, and Steiglitz (1977), On the Complexity of Local Search for the Traveling Salesman Problem. SIAM J. of Computing 6, pg 76-90, 1977.

Parzen, E. (1962), Stochastic Processes. Holder-Day Inc., San Francisco, CA, 1962.

Rabin. M.O. (1976), Probabilistic Algorithms. in Proceedings of the Symposium on New Directions and Recent Results in Algorithms and Complexity. Academic Press, 1976.

Reingold, E.M., Neivergelt, J., and Deo, N. (1977), Combinatorial Algorithms: Theory and Practice. Prentice-Hall, Englewood Cliffs, NJ, 1977.

Reiter, S., and Sherman, G.S. (1965) Discrete Optimizing. SIAM Journal 13, No.3, 1965, 864-889.

Riordan, J. (1859), An Introduction to Combinatorial Analysis. John Wiley & Sons Inc., New York, NY, 1958.

Rinnooy Kan, A.H.G. (1976), On Mittens Axioms for Branch and Bound. Operations Research 24, 6(1976), 1176-1178.

Rinnooy Kan, A.H.G. (1974), On Mittens Axioms for Branch and Bound. Working Paper W/74/45/03, Graduate School of Management, Delft, The Netherlands, 1974.

Salkin, H.M., and De Kluyver, C.A. (1975), The Knapsack Problem: A Survey. Naval Research Logistics Quarterly 22, pg 127-144, 1975.

Shapiro, D.M. (1966), Algorithms for the Solution of the Optimal cost and Bottleneck Traveling Salesman Problems. Unpublished Sc.D. Thesis, Washington University, St. Louis, Missouri, 1966.

Smith, T.H.C., Srinivasan, V., and Thompson, G.L. (1977), Computational Performance of Three Subtour Elimination Algorithms for solving Traveling Salesman Problems. Annals of Discrete Mathematics, Vol. 1, 1977, 495-506.

Weiner, P., Savage, S.L., and Bagchi, A. (1973), Neighborhood Search Algorithms for Finding Optimal Traveling Salesman Tours must be Inefficient. Proc. of the Fifth STOC Conference, 1973,

207-213.

INITIAL DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, VA 22314	2
Library Code 0142 Naval Postgraduate School Monterey, CA 93940	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93940	1
Assistant Professor Douglas R. Smith Code 52Sc Naval Postgraduate School Monterey, CA 93940	25
National Science Foundation Washington, D. C. 20550	2