State-Space Relaxation Procedures for the Computation of Bounds to Routing Problems

Nicos Christofides
Imperial College, London SW7, 2BX

A. Mingozzi
Sogesta, Urbino, Italy

P. Toth
University of Bologna, Italy

It is well-known that few combinatorial optimization problems can be solved effectively by dynamic programming alone, since the number of vertices of the state space graph is enormous. What we are proposing here is a general relaxation procedure whereby the state-space associated with a given dynamic programming recursion is relaxed in such a way that the solution to the relaxed recursion provides a bound which could be embedded in general branch and bound schemes for the solution of the problem. This *state space relaxation* method is analogous to Langrangian relaxation in integer programming. This paper gives a survey of this new methodology, and gives, as examples, applications to the traveling salesman problem (TSP), the time-constrained TSP and the vehicle routing problem (VRP). Valid state space relaxations are discussed for these problems and several bounds are derived in each case. Subgradient optimization and "state space ascent" are discussed as methods of maximizing the resulting lower bounds. More details of the procedures surveyed in this paper can be found in [2, 3, 4].

I. INTRODUCTION

Consider a graph $G = (X, A)$, where $X \equiv \{x_1, x_2, \ldots, x_n\}$ is a set of vertices of G, A is a set of arcs, and $[c_{ij}]$ is a matrix giving the costs of these arcs. We use $c(x_i, x_j)$ and c_{ij} interchangeably. In routing problems, in general, it is required to find a route, or set of routes, in the graph G, so that certain constraints are satisfied and the total cost of the routes is minimized.

One of the most successful methods of solving routing problems is by the use of branch and bound algorithms where the effectiveness of the bounds is the most important parameter that determines the efficiency of the complete algorithm. A general methodology for computing lower bounds is Langrangian relaxation [6, 10], and although it is only one of the several different bounding schemes that are possible, it

NETWORKS, Vol. 11 (1981) 145-164
© 1981 John Wiley & Sons, Inc.
CCC 0028-3045/81/020145-20$02.00

has performed well on many different types of routing and other problems. In this paper we consider some archtypical routing problems: the traveling salesman problem (TSP), the time-constrained TSP and the vehicle routing problem (VRP). Langrangian relaxation procedures involve in the first instance the explicit formulation of a problem as an integer (or mixed-integer) program. Routing problems, in general, require solutions that represent (i) a connected graph and (ii) a traversible graph. (A connected graph is one for which there is a path from any vertex to any other vertex using only arcs of the graph. A traversible graph is one which can be drawn on paper by a line starting and finishing at the same location and passing along every arc exactly once.) Thus, explicit routing problem formulations contain (amongst others):

(i) A set of constraints to impose the connectivity of routes.

(ii) A set of constraints to impose the traversibility of the solution.

In addition-depending on the problem-constraints are required in order to: impose vehicle capacity constraints in the VRP, impose restrictions in a time constrained TSP, etc. These constraints tend to destroy whatever structure the original unconstrained problem had.

In order to overcome these difficulties, an alternative methodology has been developed to deal with routing problems. This methodology is based on the twin observations that (i) every "routing" problem is essentially a shortest path problem on some underlying graph with additional constraints, and (ii) dynamic programming is a general procedure for solving shortest path problems subject to constraints, by introducing the constraints into additional state variables in the state vector, and solving an essentially unconstrained shortest path problem on an expanded "state-space graph."

Thus, it is quite natural to look at dynamic programming as a basis for solving such problems. Consider, for example, the well-known traveling salesman problem (TSP) where a shortest route is required passing through every vertex of G once and once only. Let $f(S, x_i)$ be the least cost of a path starting at vertex x_1, passing through every vertex of S and finishing at vertex $x_i \in S$. A dynamic programming recursion giving $f(S, x_i)$ is

$$f(S, x_i) = \min_{x_j \in S - x_i} [f(S - x_i, x_j) + c_{ji}] \tag{1}$$

$\forall S \subseteq X' \equiv X - \{x_1\}$, $\forall x_i \in S$, and is initialized by $f(\{x_i\}, x_i) = c_{1i}$, $\forall x_i$. The optimum solution to the problem is then given by:

$$\min_{x_i \in X'} [f(X', x_i) + c_{i1}]$$

Recursion (1) gives a shortest path procedure on the state-space graph whose vertices correspond to the states (S, x_i) and whose arcs represent transitions from one state to another.

It is well-known [1] that few combinatorial optimization problems can be solved effectively by dynamic programming alone, since the number of vertices of the state space graph is enormous. What we are proposing here is a general relaxation procedure, whereby the state-space associated with a given dynamic programming recursion is relaxed (i.e., the number of states reduced) in such a way that the solution to the

relaxed recursion provides a bound (lower bound in the case of minimization, upper bound in the case of maximization) to the value of the true optimum. Such a relaxation could then provide bounds for embedding in general branch and bound schemes for the solution of the problem.

This *state space relaxation* method is analogous to Langrangian relaxation in integer programming. Constraints in integer programming formulations appear as state variables in dynamic programming recursions and hence constraint relaxation corresponds to state space relaxation.

This paper gives a survey of this new methodology, and gives, as examples, applications to the TSP, the time-constrained TSP, and the vehicle routing problem (VRP). Valid state space relaxations are discussed for these problems and several bounds are derived in each case. Subgradient optimization and "state space ascent" are discussed as methods of maximizing (minimizing) the resulting lower (upper) bounds for imbedding into tree-search algorithms. More details of the procedures surveyed in this paper can be found in [2-4]. For further details on dynamic programming approximations see [7, 8, 14, 15].

II. STATE-SPACE RELAXATIONS

A. General

Consider a multistage discrete system where $s = (s^1, \ldots, s^m)$ is the state variable. Let the stage be represented by i, and let $f_{0,i}(s_0, s_i)$ be the least cost of changing the state of the system from s_0 at stage 0 to s_i at stage i. The general forward dynamic programming recursion for such a system is:

$$f_{0,i}(s_0, s_i) = \underset{s_{i-1} \in \Delta^{-1}(s_i)}{\text{OPT}} [f_{0,i-1}(s_0, s_{i-1}) + v_i(s_{i-1}, s_i)], \tag{2}$$

where $\Delta(s_{i-1})$ is the set of all possible states s_i at stage i that can result from state s_{i-1} at stage $i-1$, and $\Delta^{-1}(s_i) = \{s_{i-1} | \Delta(s_{i-1}) \ni s_i\}$. $v_i(s_{i-1}, s_i)$ is the cost of changing the system from state s_{i-1} to state s_i at stage i and $v_1(s_0, s_1) = \infty$ if $s_1 \notin \Delta(s_0)$.

The usual computational problem that one encounters in the direct application of recursion (2) is the problem of dimensionality of state variable s. This is particularly true for combinatorial optimization problems where the state variable often refers to combinations of items, i.e., involves all subsets of a given set.

In the present paper we are not concerned with the exact solution of this recursion but we wish to make use of associated recursions based on various relaxations of the state variable s in order to obtain bounds which can then be used in tree search procedures for the solution of the problem.

Let $g(.)$ be a mapping from the state space S to a space G with smaller cardinality, and let $F^{-1}(g(s_i))$ be a set satisfying the condition:

$$\text{if} \quad s_{i-1} \in \Delta^{-1}(s_i) \quad \text{then} \quad g(s_{i-1}) \in F^{-1}(g(s_i)). \tag{3}$$

Recursion (2) can now be written as:

$$f_{0,i}(g(s_0), g(s_i)) = \underset{t \in F^{-1}(g(s_i))}{\text{OPT}} [f_{0,i-1}(g(s_0), t) + \bar{v}_i(t, g(s_i))], \tag{4}$$

where

$$\bar{v}_i(t, g(s_i)) = \text{OPT } [v_i(p, s^*) | g(p) = t, g(s_i) = g(s^*)]. \tag{5}$$

From the above it is clear that:

$$f_{0,i}(g(s_0), g(s_i)) \leqslant f_{0,i}(s_0, s_i) \tag{6}$$

for all i, s_0 and s_i. Note that (6) above can produce bounds which can be embedded into any tree search or other procedure for solving the original problem.

In general, the state-space relaxation which produced recursion (4) is useful only if the following conditions are satisfied:

(i) The function $g(.)$ must be such that set $F^{-1}(.)$ can be computed easily from (3) above.
(ii) The function $g(.)$ must be such that the optimization in (5) is over a small domain or a good lower bound on $\bar{v}_i(t, g(s_i))$ can be otherwise obtained.

We will illustrate some possible useful forms of the function $g(.)$ by reference to the traveling salesman problem (TSP). We note, however, that the general technique is applicable to many other combinatorial optimization problems.

We must also note here, that the effectiveness, or otherwise, of a state-space relaxation in producing bounds is relative to a given dynamic programming formulation. This is similar to the case in integer programming where the effectiveness of Lagrangian relaxation in producing bounds is relative to the integer programming formulation. Moreover, continuing this analogy, redundant state-space definitions can lead to better state-space relaxations and hence better bounds just as in integer programming the addition of redundant constraints can lead to better Lagrangian relaxations and hence better bounds. (See Sec. V A).

B. State-Space Relaxations for the TSP

Consider the dynamic programming formulation of the TSP given by recursions (1) in Sec. I. The state variable s in that formulation is (S, x), and the stage is the cardinality of S. Let $g(.)$ be the mapping function from the domain of (S, x) to some other vector space $(g(S), x)$.

For the TSP we have:

$$\Delta^{-1}(S, x) = \{(S - x, y) | y \in S - x\},$$

$$F^{-1}(g(S), x) = \{(g(S - x), y) | y \in S - x\} \tag{7}$$

$$\subseteq \{(g(S - x), y) | y \in \Gamma^{-1}(x)\}. \tag{8}$$

Note that since we are merely interested in lower bounds to the TSP, $F^{-1}(g(S), x)$ in (7) may be replaced by any larger set that is easier to compute. One such choice is $\Gamma^{-1}(x)$, it may, however, be possible to use a smaller set. This can be achieved, if, for example, it is possible to deduce that no set $(S - x)$ producing $g(S - x)$ can contain a

certain vertex y, in which case the state $(g(S - x), y)$ can be removed from the definition of $F^{-1}(.,.)$ by expression (8).

In general, $F^{-1}(g(S), x)$ can be defined by an expression of the form

$$F^{-1}(g(S), x) = \{(g(S - x), y) | y \in E^{-1}(g(S), x)\}$$

where

$$\Gamma^{-1}(x) \supseteq E^{-1}(g(S), x) \supseteq S - x.$$

Note that the optimization in (5) for the TSP is independent of $g(.)$ and depends only on x and y. Thus condition (ii) of Sec. II A is satisfied. Recursion (4) for the TSP now becomes:

$$f(g(S), x) = \min_{(g(S-x), y) \in F^{-1}(g(S), x)} [f(g(S - x), y) + c(y, x)]. \tag{9}$$

Condition (i) of Sec. II A is satisfied if $g(.)$ is chosen to be a *separable function*, so that given $g(S)$ and x, $g(S - x)$ can be computed. In this case the minimization of (9) is only over the variable $y \in E^{-1}(g(S), x)$ and (9) becomes:

$$f(g(S), x) = \min_{y \in E^{-1}(g(S), x)} [f(g(S - x, y)) + c(y, x)]. \tag{10}$$

With the initialization:

$$\begin{aligned} f(w, y) &= c(x_1, y), && \text{if } w = g(\{y\}) \\ &= \infty, && \text{otherwise.} \end{aligned}$$

By using the transpose of the cost matrix $[c_{ij}]$ we can define a second "reverse" function $f'(g(S), x)$ by a recursion exactly analogous to (10). $f'(g(S), x)$ corresponds to paths starting from state $(g(S), x)$ and finishing at the final state $(g(X), x_1)$. For symmetric TSP's $f'(.) = f(.)$.

Note that backtracking can produce the solutions corresponding to $f(.,.)$ and $f'(.,.)$.

C. Forms of Mapping Function $g(.)$ for the TSP

We have derived in the previous section that $g(.)$ can be any separable function. In this section we examine a variety of such functions:

(A) $g(S) = |S|$. (Cardinality relaxation: n-path).

Let $k = |S|$. We then have $g(S - x_i) = g(S) - 1$. Recursion (10) becomes:

$$f(k, x_i) = \min_{x_j \in E^{-1}(k, x_i)} [f(k - 1, x_j) + c(x_j, x_i)] \tag{11}$$

and is initialized by $f(1, x_i) = c(x_1, x_i)$. In (11) $E^{-1}(k, x_i) = \Gamma^{-1}(x_i)$. This recursion is the shortest n-path relaxation of the TSP.

(B) $g(S) = \Sigma_{x_i \in S} q_i$. ($q$-path relaxation).

Let us associate an integer number $q_i \geqslant 1$ with every vertex $x_i \in X$, ($q_1 = 0$) and define $g(S) = q \equiv \Sigma_{x_i \in S} q_i$. We then have:

$$g(S - x_i) = g(S) - q_i$$

Recursion (10) now becomes:

$$f(q, x_i) = \min_{x_j \in E^{-1}(q, x_i)} [f(q - q_i, x_j) + c(x_j, x_i)] \tag{12}$$

and is initialized by:

$$\begin{aligned} f(q, x_i) &= c(x_1, x_i), && \text{if } q = q_i \\ &= \infty, && \text{if } q \neq q_i. \end{aligned}$$

In (12) $E^{-1}(q, x_i)$ can be defined as:

$$E^{-1}(q, x_i) = \{x_j | \Gamma(x_j) \ni x_i, q - q_i \geqslant q_j\}.$$

This recursion is the shortest q-path relaxation.

(C) $g(S) = (|S|, \Sigma_{x_i \in S} q_i)$. ($(n - q)$-path relaxation).

Let us associate an integer number $q_i \geqslant 1$ with every vertex $x_i \in X$ ($q_1 = 0$) to define $g(S)$ as the vector $(k, q) = (|S|, \Sigma_{x_i \in S} q_i)$. We then have: $g(S - x_i) = (k - 1, q - q_i)$ and recursion (10) becomes:

$$f(k, q, x_i) = \min_{x_j \in E^{-1}(k, q, x_i)} [f(k - 1, q - q_i, x_j) + c(x_j, x_i)] \tag{13}$$

and is initialized by

$$\begin{aligned} f(k, q, x_i) &= c(x_1, x_i), && \text{if } q = q_i \text{ and } k = 1 \\ &= \infty, && \text{otherwise.} \end{aligned}$$

In (13) $E^{-1}(k, q, x_i)$ can be defined as

$$\{x_j | \Gamma(x_j) \ni x_i, q - q_i \geqslant q_j\}.$$

(D) $g(S) = (\Sigma_{x_i \in S} q_i^1, \Sigma_{x_i \in S} q_i^2)$.

If we associate two integer numbers q_i^1, q_i^2 (with $q_i^1 + q_i^2 \geqslant 1$) with every vertex x_i and define $g(S)$ as the vector

$$(Q^1, Q^2) = \left(\sum_{x_i \in S} q_i^1, \sum_{x_i \in S} q_i^2 \right),$$

we then have:

$$g(S - x_i) = \left(\sum_{x_r \in S} q_r^1 - q_i^1, \sum_{x_r \in S} q_r^2 - q_i^2 \right)$$

and recursion (10) becomes:

$$f(Q^1, Q^2, x_i) = \min_{x_j \in E^{-1}(Q^1, Q^2, x_i)} [f(Q^1 - q_i^1, Q^2 - q_i^2, x_j) + c(x_j, x_i)] \qquad (14)$$

And is initialized by

$$f(Q^1, Q^2, x_i) = c(x_1, x_i), \quad \text{if } Q^1 = q_i^1 \text{ and } Q^2 = q_i^2$$
$$= \infty, \quad \text{otherwise.}$$

In (14) $E^{-1}(Q^1, Q^2, x_i)$ can be defined as:

$$E^{-1}(Q^1, Q^2, x_i) = \{x_j | \Gamma(x_j) \ni x_i, Q^1 - q_i^1 \geqslant q_j^1, Q^2 - q_i^2 \geqslant q_j^2\}$$

An obvious generalization of this case is when we associate to each vertex x_i, k (instead of just 2) integers $q_i^1, \ldots, q_i^k$. Note, however, that k cannot be much above 2 or 3 since the state space is mapped into the set of integers $(1 - L)$ and

$$L = \sum_{x_i \in X} q_i^1 \times \cdots \times \sum_{x_i \in X} q_i^k$$

can be very large.

Clearly case (A) is a special case of (B) which is a special case of (C) which is a special case of (D).

(E) $g(S) = (\Sigma_{x_i \in S_1 \cap S} q_i, \Sigma_{x_i \in S_2 \cap S} q_i, \ldots, \Sigma_{x_i \in S_r \cap S} q_i)$.

Let $S_1, \ldots, S_r$ be a partition of the set X', and let q_i be as given in case (B). Define $g(S)$ as the vector:

$$(\tilde{Q}^1, \ldots, \tilde{Q}^r) = \left(\sum_{x_i \in S_1 \cap S} q_i, \ldots, \sum_{x_i \in S_r \cap S} q_i \right).$$

Then $g(S - x_i) = (\tilde{Q}^1, \ldots, \tilde{Q}^l - q_i, \ldots, \tilde{Q}^r)$, where $x_i \in S_l$.

Recursion (10) becomes:

$$f(\tilde{Q}^1, \ldots, \tilde{Q}^r, x_i) = \min_{x_j \in E^{-1}(\tilde{Q}^1, \ldots, \tilde{Q}^r, x_i)} [f(\tilde{Q}^1, \ldots, \tilde{Q}^l - q_i, \ldots, \tilde{Q}^r, x_j) + c(x_j, x_i)]$$

(15)

and is initialized by

$$f(\tilde{Q}^1, \ldots, \tilde{Q}^r, x_i) = c(x_1, x_i), \quad \text{if } \tilde{Q}^l = q_i \text{ and } \tilde{Q}^p = 0, p \neq l$$
$$= \infty, \quad \text{otherwise.}$$

In (15) $E^{-1}(\tilde{Q}^1, \ldots, \tilde{Q}^r, x_i)$ can be defined as

$$\{x_j | \Gamma(x_j) \ni x_i, \tilde{Q}^w \geqslant q_j, \text{ where } x_j \in S_w\}.$$

The state space is now mapped onto the set of integers $(1 - L)$ where

$$L = \sum_{x_i \in S_1} q_i \times \cdots \times \sum_{x_i \in S_r} q_i.$$

Clearly this case corresponds to case (B) given earlier and can easily be generalized for cases (C) and (D).

D. Imposing Additional Conditions

In Sec. II B we have seen how a mapping function $g(.)$ can be used to reduce the dimensionality of state space and in Sec. II C we introduced various forms of this function. The introduction of $g(.)$ does not, in general, allow any detailed knowledge of the state and hence one cannot impose additional conditions to ensure that a feasible solution to the original problem is obtained. However, certain specific restrictions can be imposed (for any arbitrary $g(.)$) without increasing the dimensionality of state space and these restrictions improve the quality of the solution generated by solving the relaxed problem. In the case of the TSP relaxation defined by recursion (10), for example, it is possible to impose the condition that the path should not contain loops formed by 3 consecutive vertices, i.e., to avoid paths of the form $\ldots x_i, x_j, x_i, \ldots$. This can be done in the following way.

Let $p(g(S), x)$ be the vertex just prior to x on the path corresponding to $f(g(S), x)$. Let $\phi(g(S), x)$, be the least cost path from the initial state $(g(\{x_1\}), x_1)$ to state $(g(S), x)$ and with $\pi(g(S), x) \neq p(g(S), x)$ where $\pi(g(S), x)$ is the vertex just prior to vertex x on the path corresponding to $\phi(g(S), x)$.

Recursion (10) now becomes:

$$f(g(S), x) = \min_{y \in E^{-1}(g(S), x)} [f(g(S - x), y) + c(y, x)], \quad \text{if } p(g(S - x), y) \neq x$$
$$= \min_{y \in E^{-1}(g(S), x)} [\phi(g(S - x), y) + c(y, x)], \quad \text{otherwise.} \tag{16}$$

The value of y producing the above minimum is $p(g(S), x)$.

Identical recursions can be written for $\phi(g(S), x)$ but with the restriction that the minimizations are over those $y \neq p(g(S), x)$. The value of y producing the minimum of the expressions for $\phi(g(S), x)$ is $\pi(g(S), x)$.

The initialization is now

$$f(w,y) = c(x_1,y) \quad \text{and} \quad p(w,y) = x_1, \qquad \text{if } w = g(\{y\})$$
$$= \infty, \qquad \text{otherwise}$$

and

$$\phi(w,y) = \infty.$$

Similar recursions are given for the specific functions $g(.)$ in [4] and [13].

III. BOUNDS FROM STATE-SPACE RELAXATIONS

It is clear from (6) that the state space relaxations of the dynamic programming recursions of combinatorial problems can be used to obtain lower bounds on the value of the solution to these problems. For the example case of the TSP we will describe how some of these bounds can be obtained. We should note, however, that this is by no means an exhaustive list of bounds that can be derived from the state space relaxation of the TSP. Bounds for other routing problems are discussed in the next sections.

A simple bound can be obtained from recursion (11) by noting that $f(n-1,x_i)$ is the least cost path (n-path) starting from vertex x_1, finishing at vertex x_i and having $(n-1)$ arcs. The bound is then:

$$B_1 = \min_{x_i \in \Gamma^{-1}(x_1)} [h(x_i) + c(x_i,x_1)], \tag{17}$$

where $h(x_i) = f(n-1,x_i)$. This bound was obtained from different considerations and used for the TSP by Houck et al. [13].

Similar bounds B_2 to B_5 can be obtained from recursion (12) to (15) and are given by eq. (17), where

$$\text{for } B_2 : h(x_i) = f(\bar{Q}, x_i), \quad \text{where} \quad \bar{Q} = \sum_{x_i \in X} q_i.$$

$$\text{For } B_3 : h(x_i) = f(n-1, \bar{Q}, x_i).$$

$$\text{For } B_4 : h(x_i) = f(\bar{Q}^1, \bar{Q}^2, x_i),$$

where

$$\bar{Q}^1 = \sum_{x_i \in X} q_i^1 \quad \text{and} \quad \bar{Q}^2 = \sum_{x_i \in X} q_i^2.$$

$$\text{For } B_5 : h(x_i) = f(\hat{Q}^1, \ldots, \hat{Q}^r, x_i),$$

where

$$\hat{Q}^l = \sum_{x_i \in S_l} q_i, l = 1, \ldots, r.$$

A. Bounds From "Through-Circuits"

Let us now define a function $\psi(k, x_i)$ as the least cost of a circuit of cardinality n starting and finishing at vertex x_1 and passing through vertex x_i at position k. This function can be computed as:

$$\psi(k, x_i) = f(k, x_i) + f'(n - k, x_i), \tag{18}$$

where $f'(.)$ is defined in Sec. II B.

Similarly, we define a function $\psi(q, x_i)$ as the least cost of a circuit of cardinality n starting and finishing at vertex x_1 and passing through vertex x_i, when the sum of the q_j of all vertices x_j preceding x_i along the circuit (and including x_i itself) adds up to q. We will call q the "*load-position*" of x_i. The function can be computed as

$$\psi(q, x_i) = f(q, x_i) + f'(\overline{Q} - q + q_i, x_i). \tag{19}$$

Similarly, we define a function $\psi(k, q, x_i)$ as the least cost of a circuit of cardinality n starting and finishing at vertex x_1 and passing through vertex x_i at position k, when the load-position of x_i is q. The function can be computed as:

$$\psi(k, q, x_i) = f(k, q, x_i) + f'(n - k, \overline{Q} - q + q_i, x_i). \tag{20}$$

In an analogous way we can define functions $\psi(Q^1, Q^2, x_i)$ and $\psi(\tilde{Q}^1, \ldots, \tilde{Q}^r, x_i)$. The circuits corresponding to the functions ψ are called "through-circuits." Let b_{ik} be a lower bound on the cost of the least cost tour starting and finishing at vertex x_1 and passing through vertex x_i at position k.

Restricting attention to the first two state-space relaxations, b_{ik} can be computed as:

either:
$$b_{ik} = \psi(k, x_i), \tag{21a}$$

or:
$$b_{ik} = \min_{q_i \leq q \leq \overline{Q}} [\psi(k, q, x_i)]. \tag{21b}$$

The second of the above expressions clearly dominates the first.

Let us now construct an $(n - 1) \times (n - 1)$ matrix $[b_{ik}]$ where each row corresponds to a vertex $x_i (x_i \neq x_1)$ and each column corresponds to an integer $k = 1, \ldots, n - 1$.

Every vertex must be in some position of a feasible tour and there can only be one vertex in any position. Thus, a whole family of bounds for the TSP can be derived by the use of matrix $[b_{ik}]$ as follows:

(1) The value of the solution of the bottleneck assignment problem [9] defined by $[b_{ik}]$ is a lower bound.

(2) Any lower bound to the above bottleneck assignment problem is obviously also a bound to the TSP. A simple such bound is, for example:

$$\operatorname{Max}_{x_i} \left[\min_{k=1,\ldots,n-1} b_{ik} \right].$$

In a similar way, we can define b_{iq} to be a lower bound on the cost of the least cost tour starting and finishing at x_1 and passing through vertex x_i, when the load-position of x_i is q. b_{iq} can be computed as:

$$\textit{either:} \qquad b_{iq} = \psi(q, x_i) \tag{22a}$$

$$\textit{or:} \qquad b_{iq} = \min_{k=1,\ldots,n-1} [\psi(k, q, x_i)] \tag{22b}$$

Clearly the second of the above expressions dominates the first.

From similar considerations to those above, the value of the solution of the rectangular bottleneck assignment problem [9] defined by the matrix $[b_{iq}]$ is a lower bound to the TSP. Once more, any simple lower bound to the above assignment problem is also a bound to the TSP.

IV. TSP WITH TIME CONSTRAINTS

Consider the TSP defined on the graph $G = (X, A)$ with the following additional restrictions. With each vertex $x_i \in X$ we associate r_i "time windows," the kth one being defined by the pair of times (e_i^k, u_i^k), where $e_i^k < u_i^k, k = 1, \ldots, r_i$. We assume (without loss of generality) that the "time windows" are disjoint and ordered so that $u_i^{k-1} < e_i^k, k = 1, \ldots, r_i, x_i \in X$, and where $u_i^0 = 0$ for all x_i.

Let δ_i be the "processing" time of vertex x_i. For the purposes of clarity we will assume that the cost matrix $[c_{ij}]$ of the TSP represents travel times. We require a TSP tour $(x_1, \ldots, x_{i_n}, x_1)$ starting from a specified vertex x_1 visiting every other vertex once only and returning to vertex x_1. The time of departure of the salesman from vertex x_1 is 0 and the time of visiting vertex x_i is t_i. A feasible tour is one which satisfies $e_i^k \leqslant t_i \leqslant u_i^k$ for some $k = 1, \ldots, r_i$, for every $x_i \in X$.

If the salesman travels from vertex x_j to vertex x_i directly, the visiting time t_i is given by:

$$\left.\begin{array}{lll} \textit{either:} & t_i = t_j + \delta_j + c_{ji}, & \text{if } e_i^k \leqslant t_j + \delta_j + c_{ji} \leqslant u_i^k \\ \textit{or:} & t_i = e_i^k, & \text{if } u_i^{k-1} < t_j + \delta_j + c_{ji} \leqslant e_i^k \\ \textit{or:} & t_i = \infty, & \text{if } t_j + \delta_j + c_{ji} > u_i^{r_i}. \end{array}\right\} \text{for some } k = 1, \ldots, r_i \tag{23}$$

We wish to find that tour $(x_1, \ldots, x_{i_n}, x_1)$ which minimizes the time, T say, of returning to vertex x_1; i.e., we wish to minimize $T \equiv t_{i_n} + \delta_{i_n} + c_{i_n 1}$.

Note that t_i as defined in eq. (23) above implies "waiting" by the salesman if he arrives at a vertex outside one of its time windows.

A. Dynamic Programming Formulation

Let $R(x_i)$ be the set of vertices from which it is possible to go directly to vertex x_i. In the worst case we can take $R(x_i) = X - \{x_i\}$. However, in many cases it is possible

(by means of simple *a priori* tests) to restrict $R(x_i)$ to be only a small subset of X in any optimal solution [2]. Let $f(S, x_i)$ be the least duration of a feasible path starting at x_1 visiting every vertex in the set S and finishing at vertex x_i (excluding time δ_i). For a given S and x_i let:

$$h = \min_{(S-x_i, x_j)\in\Delta^{-1}(S,x_i)} [f(S - x_i, x_j) + \delta_j + c_{ji}], \tag{24a}$$

where

$$\Delta^{-1}(S, x_i) = \{(S - x_i, y) | y \in (S - x_i) \cap R(x_i)\}.$$

We now have:

$$\left.\begin{aligned} f(S, x_i) = h, &\quad \text{if } e_i^k \leqslant h \leqslant u_i^k \\ = e_i^k, &\quad \text{if } u_i^{k-1} < h \leqslant e_i^k \end{aligned}\right\} \text{for some } k = 1, \ldots, r_i$$
$$= \infty, \quad \text{if } h > u_i^{r_i}. \tag{24b}$$

B. Relaxations and Bounds

The same mapping functions used for the state space relaxation of the pure TSP can also be used for the present problem. Let LB be a known lower bound and UB a known upper bound to the optimum solution value. Consider a given vertex x_i. The kth time window of this vertex is between the times e_i^k and u_i^k. If the salesman arrives at the vertex in this time window he will take at least time LB - $u_i^k + \delta_i$, and at most time UB - $e_i^k + \delta_i$, to arrive back to vertex x_1. Thus, if we are considering the "reverse" path, we can say that in the "reverse" direction the salesman must leave x_1 at time 0 and arrive at x_i between time $\bar{e}_i^k$ = LB - $u_i^k + \delta_i$ and time $\bar{u}_i^k$ = UB - $e_i^k + \delta_i$. Thus, if all windows $k = 1, \ldots, r_i$ are transformed in this way for every customer x_i, we can compute the "reverse" function $f'(g(S), x)$ as explained in Sec. II B. Note that the above transformation of the time windows are exact only if UB = LB = T^* (the value of the optimal solution), otherwise the transformation represents a relaxation of the time-window restrictions. As a result of this relaxation some time windows for a given customer may no longer remain disjoint. This problem, however, can be alleviated by absorbing two or more overlapping windows into a single time slot.

Having computed $f(g(S), x)$ and $f'(g(S), x)$ we can now derive from the state-space relaxation of recursions (24a) and (24b) the same bounds as those introduced for the pure TSP.

More extensive discussion of this problem, including the case where no "waiting" is allowed, and analyzing the computational performance of the resulting bounds is given in [2]. This reference also considers conditions under which the bounds coincide with the optimal answer. Other types of constrained TPS's are considered in [2].

V. THE VEHICLE ROUTING PROBLEM

We are considering a problem in which a set of geographically dispersed "customers" with known requirements must be served with a fleet of "vehicles" stationed at a

central facility or depot in such a way as to minimize some distribution objective. It is assumed that all vehicle routes must start and finish at the depot [4].

The *vehicle routing problem* (VPR) is a generic name given to a whole class of problems involving the visiting of "customers" by "vehicles." The basic VPR considered here is as follows:

A graph $G = (X, A)$ is defined by the set X of its vertices and the set A of its arcs.

Let $X' = \{x_i | i = 2, \ldots, n\}$ be used for the set of n customers and let x_1 be the depot. $X = X' \cup \{x_1\}$. A customer x_i has an associated quantity q_i of some product to be delivered by a vehicle. We assume that M identical vehicles each of capacity Q are stationed at the depot.

The number of vehicles is assumed to be large enough for a feasible solution to exist. We further assume that the "cost" of the least cost path from every vertex x_i to every vertex x_j is given as c_{ij}. It is required that the total quantity on each vehicle route is less than or equal to Q. The objective in the VPR that is considered here is to design feasible routes—one for each vehicle-in order to supply all of the customers and minimize the total cost of all the routes. For the purposes of this section the "cost" c_{ij} mentioned above can be taken to be either travel distances or travel times between the customers. The VRP defined above is a generalization of the traveling salesman problem discussed earlier.

A. Dynamic Programming Formulations

1. Formulation 1

Let $f(m, S)$ be the least cost of supplying a set S of customers using only m vehicles and let $v(S)$ be the solution to the TSP defined by the set S of customers and the depot x_1.

With the above definition, the dynamic programming recursion becomes

$$f(m, S) = \min_{L \subset S} [f(m - 1, S - L) + v(L)]$$

subject to

$$\sum_{x_i \in S} q_i - (m - 1)\, Q \leqslant \sum_{x_i \in L} q_i \leqslant Q \tag{25a}$$

for $m = 2, \ldots, M$, and where $S \subseteq X'$ must satisfy

$$\bar{Q} - (M - m)\, Q \leqslant \sum_{x_i \in S} q_i \leqslant m \cdot q \tag{25b}$$

where

$$\bar{Q} = \sum_{x_i \in X'} q_i.$$

The restrictions on sets L and S are so as to avoid the computation of $f(\cdot)$ and $v(\cdot)$ for sets that can only lead to load-infeasible completions. For $m = M$ only $S = X'$ need be considered, and the recursion is initialized by $f(1, S) = v(S)$.

2. Formulation 2

An alternative formulation which contains some state redundancy (namely parameter k), with respect to the above state definition is given below. It is worthwhile to recall here the statement in Sec. II A that such redundancy may prove to be useful after the state-space relaxation.

Let $f(m, S, k)$ be the least cost of supplying a set S of customers using only m vehicles with the last customers of the corresponding m routes being among the customers $x_2, \ldots, x_k$. Let $v(S, k)$ be the solution to the TSP through a set S of customers, where the last customer on the route is x_k. The recursion now, is:

$$f(m, S, k) = \min \left[f(m, S, k-1), \min_{L \subset S} \{ f(m-1, S-L, k-1) + v(L, k) \} \right]$$

$$\sum_{x_l \in S} q_l - (m-1)\, Q \leqslant \sum_{x_l \in L} q_l \leqslant Q. \tag{26}$$

3. Formulation 3

Yet another dynamic programming formulation of the VRP can be given as follows. Let $f(m, S, \bar{q}, x_i)$ be the least cost of supplying the set S of customers using m vehicles where the first $(m-1)$ of the routes are closed and the last route is open, has a total load of $\bar{q}$ and finishes at customer x_i ($\bar{q}$ is the load-position of x_i on route m). The dynamic programming recursion now becomes:

$$f(m, S, \bar{q}, x_i) = \min_{x_j \in S \cap \Gamma^{-1}(x_i)} [f(m, S - x_i, \bar{q} - q_i, x_i) + c_{ji}] \tag{27a}$$

for $q_i < \bar{q} \leqslant Q, m = 1, \ldots, M$

$$f(m, S, q_i, x_i) = \min_{\substack{x_j \in S \cap \Gamma^{-1}(x_1) \\ x_j \neq x_i}} [f(m-1, S - x_i, \bar{q}, x_j) + c_{j1}] + c_{1i}$$

$$\sum_{x_k \in S} q_k - q_i - (m-2)\, Q \leqslant \bar{q} \leqslant Q \tag{27b}$$

for $m = 2, \ldots, M$. For $\bar{q} < q_i$, $f(m, S, \bar{q}, x_i) = \infty$. For both recursions (27a) and (27b) $S \subseteq X'$ must satisfy conditions similar to (25b) above in order to avoid the computation of functions for those S that can have no feasible completions.

The functions are initialized by

$$f(1, \{x_i\}, \bar{q}, x_i) = c_{1i}, \quad \text{for } \bar{q} = q_i$$
$$= \infty, \quad \text{otherwise.}$$

The optimum value of the VRP solution is given by:

$$\min_{\substack{x_l \in \Gamma^{-1}(x_1) \\ \bar{q} - (M-1)Q \leqslant \bar{q} \leqslant Q}} [(f(M, X', \bar{q}, x_l) + c_{l1}]. \tag{28}$$

B. State-Space Relaxations and Bounds

The same mapping functions used for the state-space relaxation of the TSP can be used for relaxing the above recursions for the VRP. The variable S in the state vector in all DP formulations can be relaxed via the mapping $g(S)$ where:

$$g(S) = |S|, \text{ or } g(S) = \sum_{x_i \in S} q_i, \text{ or } g(S) = \left(|S|, \sum_{x_i \in S} q_i\right),$$

or:

$$g(S) = \left(\sum_{x_i \in S} q_i^1, \sum_{x_i \in S} q_i^2\right), \text{ or } g(S) = \left(\sum_{x_i \in S_1 \cap S} q_i, \ldots, \sum_{x_i \in S_r \cap S} q_i\right)$$

for some partition $S_1, \ldots, S_r$ or the vertices X', explained earlier.

1. Relaxation of Formulation 1

Let us consider $g(S) = q = \Sigma_{x_i \in S} q_i$ as an example. The relaxed recursion (25a) becomes:

$$f(m, q) = \min_{\overline{Q} - (M-1)Q \leq p \leq \min[q, Q]} [f(m - 1, q - p) + \overline{v}(p)] \tag{29}$$

where $\overline{v}(p)$ is given by an expression similar to (5) and for this case becomes the least cost of circuit C for which $\Sigma_{x_i \in C} q_i = p$. Since this is itself a hard problem, we will redefine $\overline{v}(p)$ to be, instead, a lower bound on the cost of such a circuit. One such bound is

$$\overline{v}(p) = \min_{x_i \in \Gamma^{-1}(x_1)} [f(p, x_i) + c_{i1}], \tag{30}$$

where $f(p, x_i)$ is given by eq. (12). A lower bound to the VRP is then given by: $f(M, \overline{Q})$.

2. Relaxation of Formulation 2

A better bound can be obtained from recursion (26). One such bound is obtained and used in [4] by considering a set-partitioning formulation of the VRP with redundant constraints and relaxing some of the original constraints.

Using the same mapping function $g(S)$ as above we get the relaxation of recursion (26) as:

$$f(m, q, k) = \min [f(m, q, k - 1),$$

$$\min_{\overline{Q} - (M-1)Q \leq p \leq \min[q, Q]} \{f(m - 1, q - p, k - 1) + \overline{v}(p, k)\}]. \tag{31}$$

Once more we can define $\overline{v}(p, k) = f(p, x_k) + c_{k1}$. A lower bound to the VRP is given by $f(M, \overline{Q}, n) \geq f(M, \overline{Q})$.

3. Relaxation of Formulation 3

Let us once more use mapping function $g(S) \equiv q = \Sigma_{x_i \in S} q_i$ for relaxing recursion (27). The relaxed recursion (27a) becomes:

$$f(m, q, \bar{q}, x_i) = \min_{x_j \in \Gamma^{-1}(x_i)} [f(m, q - q_i, \bar{q} - q_i, x_j) + c_{ji}]$$

and similarly for the relaxed recursion (27b). A lower bound that is obtained directly from the above recursion is:

$$\min_{\substack{x_i \in \Gamma^{-1}(x_1) \\ \bar{Q} - (M-1)Q \leq \bar{q} \leq Q}} [f(M, \bar{Q}, \bar{q}, x_i) + c_{i1}]$$

Let $\Phi(m, q, x_i)$ be a lower bound on the value of the optimum solution when x_i is on the mth route, and the sum of the q_j's for all customers x_j on routes 1 to $(m - 1)$ and those customers on route m preceding x_i (including x_i) adds up to q. As for the TSP we will call q the *cumulative load-position* of x_i. We have

$$\Phi(m, q, x_i) = \min_{q_i \leq \bar{q} \leq Q} [f(m, q, \bar{q}, x_i) + \min_{Q_m} \{f'(M - m + 1, \bar{Q} - q + q_i, Q_m - \bar{q} + q_i, x_i)\}], \quad (32)$$

where

$$\max\left[\bar{q}, \left(\frac{\bar{Q} - q + \bar{q}}{M - m + 1}\right)\right] \leq Q_m \leq \min\left[\left(\frac{q - \bar{q}}{m - 1}\right), Q\right] \quad (32)$$

and $f'(\cdot)$ is the "reverse" function corresponding to $f(\cdot)$ as mentioned in Sec. II B. Q_m is the possible total load of route m.

The conditions on Q_m are derived by arbitrarily considering the routes to be numbered $1, \ldots, m$ in descending order of their total load. Thus, $f'(\cdot)$ must consider the routes in the reverse order and hence has to be computed even if the matrix $[c_{ij}]$ is symmetric. Note that the first term in the square brackets of recursion (32) is independent of Q_m and depends only on $\bar{q}$. Thus, it is convenient for a given (m, q, x_i) to tabulate the minimum in the curly brackets of recursion (32) with respect to $\bar{q}$, before performing the external minimization of that recursion.

One possible bound can be obtained as follows: Let

$$b_{iq} = \min_m [\Phi(m, q, x_i)].$$

Since every customer can be only at one cumulative load-position, the value of the solution to the bottleneck rectangular assignment problem [9] defined by $[b_{iq}]$ is a lower bound to the optimum value of VRP.

An alternative bound can be obtained from matrix $[b_{im}]$, where

$$b_{im} = \min_{q} [\Phi(m, q, x_i)].$$

Let $\xi_{im} = 1$ if customer x_i is in route m, $\xi_{im} = 0$ otherwise. An optimum solution to the VRP must satisfy the constraints:

$$\begin{aligned} &\sum_{i=2}^{n} \xi_{im} q_i \leqslant Q, \qquad m = 1, \ldots, M \\ &\sum_{m=1}^{M} \xi_{im} = 1, \qquad i = 1, \ldots, n \end{aligned} \tag{33}$$

and hence the value of the solution to the bottleneck generalized assignment problem defined by the objective function:

$$\min: \max_{i,m} [b_{im} \xi_{im}]$$

subject to constraints (33) is a lower bound to the optimum value of the VRP. Several other bounds can be derived from different relaxations of formulation 3 and one such bound is derived and used in [4].

VI. PENALTY METHODS AND STATE-SPACE ASCENT

In Sec. II we described forms of the function $g(.)$ which produce valid state-space relaxations for routing problems, and in Sec. II C these were specialized to the TSP, as an example. In this section we describe how two different procedures can be used to increase the resulting bound further by:

(i) Using penalties in a Lagrangian fashion.
(ii) Using state-space modifications.

In either of the above two cases the general objective is to force the solution corresponding to the relaxed problem "closer" to feasibility.

We will again use the TSP as an example and consider the mapping function of Sec. II-C2.

A. Penalties

The lower bound corresponding to the state-space relaxation of Sec. II-C2 was computed in Sec. III as:

$$B(0) = \min_{x_i \in \Gamma^{-1}(x_1)} [f(\overline{Q}, x_i) + c(x_i, x_1)]. \tag{34}$$

Let $\bar{x}_i$ be the value of x_i producing the above minimum. Corresponding to $f(\overline{Q}, \bar{x}_i)$ is a path $P(0)$—such as that shown in Figure 1—and which can be obtained from $f(q, x_i)$ by backtracking. Clearly such a path can be infeasible. For example, in Figure 1

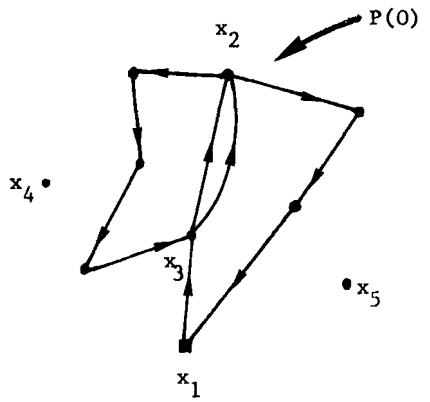

FIG. 1. Path $P(0)$ corresponding to state-space relaxation bound $B(0)$.

vertices x_4 and x_5 are not visited, whereas vertices x_2 and x_3 are visited twice. (Note that we must have $q_2 + q_3 = q_4 + q_5$).

By penalizing vertices x_2 and x_3 (by penalties λ_2 and $\lambda_3 > 0$) in a normal Lagrangian fashion, (i.e., modifying the cost as: $c'(x_j, x_i) = c(x_j, x_i) + \lambda_j + \lambda_i$), a new bound $B(\lambda)$ can be obtained by resolving recursion (12) with the cost matrix $[c'_{ij}]$ and obtain a new path $P(\lambda)$. We then wish to choose λ^* for which:

$$B(\lambda^*) = \max_{\lambda} [B(\lambda)].$$

Normal subgradient optimization methods can be used to compute λ^* [12].

B. State-Space Modifications

In choosing the mapping function $g(.)$ to be as given in Sec. II C, nothing has been said about the choice of the parameters q_i. Intuitively, it would seem desirable to choose the q_i so that the exact state-space defined by $(S, x)\ \forall\ S \subset X, x \in S$, is mapped onto the relaxed state-space as uniformly as possible. (Note that $n \cdot 2^n$ states are mapped onto $n \cdot \overline{Q}$ ($\overline{Q} = \Sigma_{x_i \in X} q_i$) states of the relaxed space). Thus, if the maximum dimensionality of the relaxed space that can be considered computationally is $\hat{Q}$, we wish to obtain: the bound:

$$B(q^*) = \max_{q} [B(q)],$$

$$\sum_{x_i \in X} q_i \leqslant \hat{Q},$$

where $B(q)$ is the bound $B(0)$ of eq. (34) for the given vector q.

The procedure for maximizing $B(q)$ in the above expression is referred to as state-space ascent [3]. This maximization is, in general, difficult since $B(q)$ is a discontinuous function of q. However, for many problems (e.g., TSP, VRP, Set Covering) simple heuristic rules for performing the maximization work quite well [2]. (See also Figure 2 of the next section).

TABLE I. Computational results for time-constrained TSP.

Number of vertices n[a]	Computing time[b]		Nodes in tree search	
	Average	Maximum	Average	Maximum
15	0.3	0.45	1	1
20	1.8	3.9	3	7
25	3.6	5.45	4	7
30	4.4	6.1	6	13
35	8.3	17.3	10	31
40	23.0	73.5	30	118
45	14.0	14.9	11	12
50	21.0	42.2	15	41

[a] For each value of n, 5 problems of that size were generated.
[b] Times are in CDC 6600 sec.

VII. COMPUTATIONAL COMMENTS

The state-space relaxation bounds described in this paper have been tested computationally on a variety of routing problems, by inserting them in branch and bound algorithms. In this section we include a short summary of the computational results obtained for the time-constrained TSP and for the VRP.

Table I shows, for a time-constrained TSP with "moderately tight" constraints, average and maximum values of computing time and number of nodes in the tree

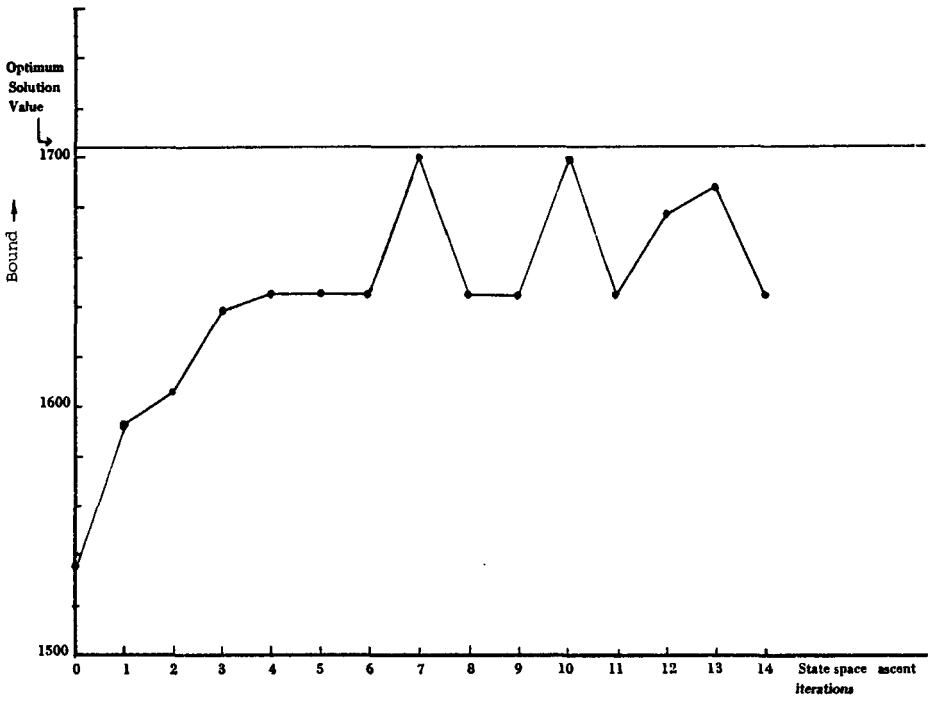

FIG. 2. State-space ascent for a 30 vertex problem (time constrained TSP).

TABLE II. Quality of state-space bounds at the root node for VRPs (relaxation of Sec. V B 2).

Number of customers	10	12	12	21	21	15	15	20	20	25
Bound % of optimum	100	93.1	98.5	98.7	95.9	96.2	95.6	99.9	96.9	99.6

search using the new bound. The small number of nodes is an indication of the effectiveness of this bound.

Figure 2 shows for a time-constrained TSP, at the root node of the search tree, the increase in the bound using a simple heuristic [2] state-space ascent for computing $B(q^*)$ in Sect. VI B. At the seventh iteration the bound is within 0.2% of the optimum.

Table II shows for ten VRP's varying in size from 10 to 25 customers, the quality of the state-space relaxation bound of Sec. V B 2 at the root node of the tree as a percentage of the value of the optimum.

References

[1] R. Bellman, *Dynamic Programming*, Princeton U. P., Princeton, 1958.

[2] N. Christofides, A. Mingozzi, and P. Toth, "Exact Algorithms for the TSP with Additional Constraints," Imperial College Internal Report No. IC, OR, 80, 23, September 1979.

[3] N. Christofides, A. Mingozzi, and P. Toth, "State-space Relaxations for Combinatorial Problems," Imperial College Internal Report No. IC, OR, 79, 09, July 1979.

[4] N. Christofides, A. Mingozzi, and P. Toth, "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Trees and Shortest Path Relaxations," *Math. Program.*, to appear.

[5] N. Christofides, "The Traveling Salesman Problem," in *Combinatorial Optimization*, N. Christofides, A. Mingozzi, P. Toth and C. Sandi, Eds., Wiley, London, 1979.

[6] M. Fisher, "Lagrangian relaxation Methods for Combinatorial Optimization," Paper presented at Summer School in Combinatorial Optimization, Urbino, Italy, 1978.

[7] B. Fox, "Finite State Approximations to Denumerable-State Dynamic Programes," *J. Math. Anal. Appl.*, **34**, 665 (1970).

[8] B. Fox, "Discretizing Dynamic Programs," *J. Opt. Theory Appl.*, **11**, 228 (1973).

[9] R. Garfinkel and G. Nemhauser, *Integer Programming*, Wiley, New York, 1970.

[10] A. Geoffrion, "Lagrangian Relaxation and its Uses in Integer Programming," *Math. Prog. Study*, **2**, 82 (1974).

[11] M. Held and R. Karp, "The TSP and Minimum Spanning Trees," *Oper. Res.*, **18**, 1138 (1970).

[12] M. Held, P. Wolfe, and H. P. Crowder, "Validation of Subgradient Optimization," *Math. Program.*, **6**, 62 (1974).

[13] D. Houck, J. C. Picard, M. Queyranne, and R. R. Vemuganti, "The Traveling Salesman Problem and Shortest n-paths," University of Maryland, 1977.

[14] T. Morin, "Computational Advances in Dynamic Programming," in *Dynamic Programming and its Applications*, M. Puterman Ed., Academic, New York, 1978.

[15] W. Whitt, "Approximations of Dynamic Programs, I & II," *Math. Oper. Res.*, **3**, 231 (1978); and **4**, 179 (1979).