APPROXIMATION ALGORITHMS FOR SOME ROUTING PROBLEMS

Greg N. Frederickson
Matthew S. Hecht*
Chul E. Kim+

Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

Several polynomial time approximation algorithms for some NP-complete routing problems are presented, and the worst-case ratios of the cost of the obtained route to that of an optimal are determined. A mixed-strategy heuristic with a bound of 9/5 is presented for the Stacker-Crane problem (a modified Traveling Salesman problem). A tour-splitting heuristic is given for k-person variants of the Traveling Salesman problem, the Chinese Postman problem, and the Stacker-Crane problem, for which a minimax solution is sought. This heuristic has a bound of e + 1 - 1/k, where e is the bound for the corresponding 1-person algorithm.

CR Categories: 5.25, 5.39, 8.3 .

Key Words and Phrases: NP-complete problems, polynomial-time approximation algorithm, heuristic, worst-case performance bound, Traveling Salesman problem, Chinese Postman problem, Stacker-Crane problem, k-person routing.

1. Introduction

The Stacker-Crane problem, brought to our attention by Rosenkrantz [12], is a modified traveling salesman problem which requires that a set of arcs be traversed, rather than a set of vertices visited. The problem encompasses practical applications such as operating a crane or a forklift, or driving a pick-up and delivery truck. The crane must start from an initial position, perform a set of moves, and return to a terminal position. We seek to schedule the moves so as to minimize total tour length. No order is imposed on the moves, and no moves may be combined and performed simultaneously.

SCP (Stacker-Crane Problem):
Let $G = (V,E,A)$ be a mixed graph with a distinguished initial vertex v_s. Let c be a cost function from $E \cup A$ to the set of nonnegative integers, with the cost of an arc never less than the cost of an edge between the same two vertices. The optimization version is to find a tour starting at v_s and traversing each arc in A, such that the cost of the tour is minimum. The recognition version is, given a positive integer C, decide if there is a tour of cost at most C.

We also consider several k-person routing problems: the k-Traveling Salesman (k-TSP), the k-Chinese Postman (k-CPP), and the k-Stacker-Cranes (k-SCP), for $k \geq 2$. These problems reflect more of the flavor of real world problems than 1-person problems. When one salesman cannot handle a large territory, k salesmen are dispatched to collectively visit each city in the territory, under the constraint that no one of the k salesmen has too large of a task.

We thus choose as our optimization criterion the minimizing of the maximum of the k salesmen's tour costs. This criterion differs from the minimizing of total tour costs subject to each salesman visiting at least one city, as suggested by Bellmore and Hong [1], and also differs from minimizing total tour costs subject to no salesman visiting more than p cities, as suggested by Miller, Tucker, and Zemlin [10].

k-TSP (k-Traveling Salesman Problem, $k \geq 2$):
Let $G = (V,E)$ be an undirected complete graph with a distinguished initial vertex v_s. A nonnegative integer cost function is defined on E that satisfies the triangle inequality. A k-tour is a set of k cycles that start from v_s and collectively visit every vertex.

k-CPP (k-Chinese Postman Problem, $k \geq 2$):
Let $G = (V,E)$ be an undirected multigraph with a cost function defined on E. A k-route is a set of k cycles that start from v_s, and collectively cover every edge in the graph.

k-SCP (k-Stacker-Cranes Problem, $k \geq 2$):
Let $G = (V,E,A)$ be a mixed graph with a cost function defined on $E \cup A$. A k-tour is a set of k-cycles that start from v_s and collectively traverse every arc in the graph.

We define the cost of a k-tour as the maximum of

*This research was partially supported by the National Science Foundation under Grant DCR-08361.
+This research was partially supported by a University of Maryland General Research Board Award.

the costs of each cycle, and an optimal k-tour is a k-tour with minimum cost. Similar definitions apply to k-route. The optimization version is thus a minimax problem and requires that given an integer $k \geq 2$, we find an optimal k-tour. The recognition version is to decide, given a positive integer C, whether a k-tour of cost at most C exists.

The recognition versions of the problems we consider in this paper are all NP-complete in the sense of Cook [3] and Karp [9]. It is well-known that TSP is NP-complete, and TSP can be transformed into SCP. Although the Chinese Postman Problem has been shown by Edmonds and Johnson [4] to be efficiently solvable for one postman, for $k \geq 2$ we show that k-CPP is no easier than k-TSP and k-SCP, all of which are NP-complete.

It has been conjectured that there exist no efficient exact algorithms for any of the optimization versions of the NP-complete problems. Consequently, algorithms have been developed and analyzed that solve certain NP-complete problems efficiently but only approximately [6,7,8,13].

Previous work on approximation algorithms for routing problems has centered on the traveling salesman problem. Sahni and Gonzalez [15] have shown that if the triangle inequality is not satisfied, the problem of finding a TSP approximate solution within a constant bound ratio of the optimal is as difficult as finding an exact solution. Papadimitriou and Stieglitz [11] have derived a similar result for local search algorithms.

If the triangle inequality is satisfied (or, equivalently, if the tour is allowed to visit vertices more than once), then approximation algorithms with a constant worst case bound exist. Rosenkrantz, Lewis, and Stearns [13] have applied worst case analysis to several incremental (insertion) heuristics. Their best worst case bound, 2, is no better than that yielded by doubling up the edges in a minimum spanning tree. Recently, Christofides [2] has developed an algorithm with a worst case bound of 1.5, which involves forming a minimum spanning tree and performing a minimum cost matching on the vertices of odd degree. We make use of Christofides' techniques in achieving a worst case bound of 1.8 for the Stacker-Crane problem.

We have found no worst case analysis for k-person routing problems. Our results indicate that generalizations of 1-person incremental algorithms may not do well. The best bounds we have been able to achieve for incremental algorithms for k-TSP have a multiplicative factor of k. In contrast, we have found that a simple tour-splitting heuristic, where a reasonable tour for 1 person is split into k tours, has better worst case behavior. The bound increases only very modestly as a function of k, so that if e is the bound on a 1-person algorithm, then the bound for our k-person TSP, CPP, and SCP algorithms is $e + 1 - 1/k$.

We now proceed to some basic definitions. An undirected graph $G = (V,E)$ consists of a set V of vertices and a set E of undirected edges. Each edge (u,v) connects two vertices u and v in V. A mixed graph $G = (V,E,A)$ consists of a set V of vertices, a set E of undirected edges, and a set A of arcs. An arc $\langle u,v \rangle$ is a directed edge from u to v, both vertices in V. We say that u is the tail of the arc $\langle u,v \rangle$, and v is the head of the arc.

A multiset is a set in which separate elements may be identical. A multigraph is a graph $G = (G,E)$ in which E is a multiset.

The degree of a vertex is the number of edges and arcs incident on the vertex. The indegree is the number of arcs directed into the vertex. The outdegree is the number of arcs directed out of the vertex. A vertex is of even degree if the degree is an even number, and is of odd degree otherwise.

Given a graph $G = (V,E)$ a path is a sequence of vertices. The first vertex in a path is called the initial vertex, the last is the terminal vertex. A cycle is a path with identical initial and terminal vertices. A tour is a cycle visiting all vertices in V, and a subtour is a tour of a subgraph. A k-tour is a set of k subtours, each visiting the initial vertex, and collectively visiting all vertices in V.

Given a multigraph $G = (V,E)$, a route is a sequence of alternating vertices and edges, covering all edges. A subroute is a route of a subgraph. Given a mixed graph $G = (V,E,A)$, we traverse a multiset $E' \cup A$ by starting at an initial vertex and proceeding along each edge or arc, in such a way that all are covered.

A matching $E' \subseteq E$ of a graph $G = (V,E)$ is a pairing of all the vertices of V. A minimum-cost matching of G is a matching such that the total cost of the edges between the paired vertices is minimum. A spanning tree of a connected graph $G = (V,E)$ is a subgraph $T = (V,E')$ of G which is a tree. A minimum cost spanning tree is a spanning tree such that the total cost of the edges E' is minimum.

2. The Stacker-Crane Problem

We distinguish three formulations of the stacker-crane problem by specifying where the path will terminate. For the first version, SCP, we require the path to be a tour and thus return to its initial vertex. For SCP2, we require the path to stop at a particular point, not necessarily the initial vertex. For SCP3, we allow the path to stop at any vertex.

2.1 NP-Completeness of the Stacker-Crane Problem

In this section we show that whether Stacker-Crane is viewed as a path or a tour problem, it is still NP-complete.

Theorem 1. The recognition version of SCP is NP-complete.

Proof. SCP is in NP, since we can nondeterministically select a tour and verify if it has cost at most C in polynomial time.

Given an instance of TSP, we can transform it in polynomial time into an instance of SCP as follows:

For each vertex v_i in the TSP graph, create two vertices v_{it} and v_{ih}, and an arc $\langle v_{it},v_{ih}\rangle$ of cost zero. For each edge (v_i,v_j), create edges (v_{it},v_{jt}), (v_{it},v_{jh}), (v_{ih},v_{jt}), and (v_{ih},v_{jh}), and assign cost $c(v_i,v_j)$ to each of the edges. (See Figure 1 for an example of the translation.) Choose any vertex v_{it} and designate it the initial vertex.

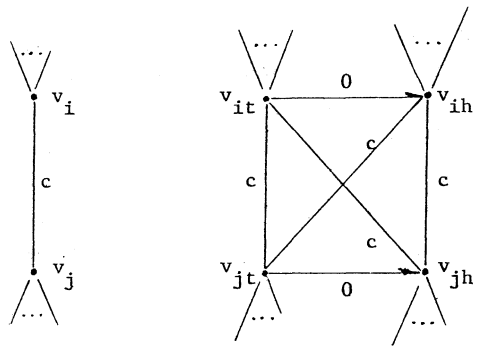

Figure 1 - Translation from TSP to SCP

A tour of cost at most C for the TSP problem can be translated into a tour of cost at most C for the SCP problem. Just replace each v_i in the TSP tour by "v_{it},v_{ih}" to get an SCP tour. Conversely, a tour of cost at most C for the SCP problem can be translated into a tour of cost at most C for the TSP problem by changing v_{ih} to v_i and deleting v_{it}, for all v_i in V. []

Corollary 1. The recognition version of SCP2 is NP-complete.

Proof. It is obvious that SCP2 is in NP. Given an instance of SCP, we can reduce it to an instance of SCP2 by designating the terminal vertex as the initial vertex. []

Corollary 2. The recognition version of SCP3 is NP-complete.

Proof. SCP3 is obviously in NP. Given an instance of SCP2, we can reduce it to an instance of SCP3 as follows:

Create a vertex v_t and an arc $\langle v_f,v_t\rangle$ from the terminal vertex v_f to the new vertex, with cost C+1. Introduce edges from v_t to each vertex v_i in V, each with cost C+1. Set the value of C for the SCP3 problem to be 2C+1.

Clearly, for an SCP3 path to have cost at most 2C+1, the path must terminate at v_t, with v_f the last vertex visited before v_t. An SCP3 path of cost at most 2C+1 can be translated into an SCP2 path of cost at most C by deleting v_t. Conversely, an SCP2 path of cost at most C can be translated into an SCP3 path of cost at most 2C+1 by attaching v_t to the end of the SCP2 path. []

2.2 An Approximation Algorithm for the Stacker-Crane Problem

We shall now consider an approximation algorithm for SCP. We point out that there is a fairly simple algorithm which consists of forming the analogue of a minimum-cost spanning tree for the arcs, and then introducing an additional edge for each edge and arc in the spanning tree. This algorithm will have a worst case bound of 2, which is approachable. In this section we shall present an algorithm with a better worst case bound.

We require that a mixed graph satisfy the following properties:

1. Each vertex is either the head or the tail of exactly one arc in A.

2. The cost of an arc between two vertices is not less than the cost of an edge between the vertices, i.e., $c\langle u,v\rangle \geq c(u,v)$.

3. The cost function on edges satisfies the triangle inequality.

If a graph does not satisfy properties 1 and 3, Algorithm PREPROCESS will transform the graph into an equivalent graph which satisfies these properties. An example of step 2 in Algorithm PREPROCESS is shown in

Figure 2. The preprocessing will not affect the cost of a stacker-crane tour, since the distance between the heads and tails of the arcs is unchanged.

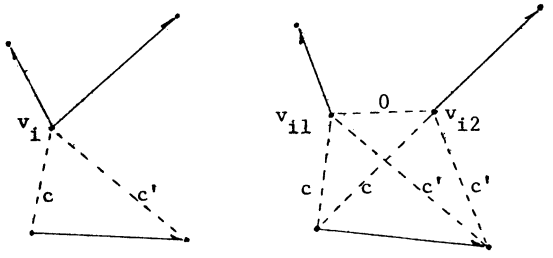

Figure 2 - Vertex Splitting of PREPROCESS

Algorithm PREPROCESS:

Input: A mixed graph G satisfying property 2.

Output: A mixed graph G' satisfying properties 1, 2, and 3.

1. If v_s is not an end point of any arc, create a terminal vertex v_f and an arc $\langle v_f, v_s \rangle$ of cost zero from the terminal vertex to the initial vertex. For all v_i in V, create edges (v_i, v_f) of cost $c(v_i, v_s)$.

2. For each vertex v_i which is shared by t arcs, replace v_i by t new vertices v_{i1}, v_{i2}, ..., v_{it}. For each pair of new vertices, set $c(v_{ik}, v_{jm})$ to zero if $i = j$ and to $c(v_i, v_j)$ otherwise.

3. Calculate the shortest path between every pair of vertices v_i and v_j, and set $c(v_i, v_j)$ to the cost obtained. []

We shall use the following notation in analyzing the worst case behavior of our algorithm. Let C^* represent the cost of an optimal tour. Let C_A be the total cost of all arcs, and let C_E^* be the cost of the edges in an optimal tour. We note that $C^* = C_A + C_E^*$.

We consider two strategies which depend on the relative size of C_A and C_E^*. If C_A is large relative to C_E^*, then the analogue of a minimum spanning tree for the arcs will be small. We can thus perform a minimum cost matching on the heads and tails of arcs, and link the resulting cycles together with the spanning edges. If C_E^* is large relative to C_A, then we essentially have a traveling salesman problem. We shrink the arcs to nodes, perform Christofides' algorithm [2], and add certain edges corresponding to the arcs to make the tour traversable.

Thus our algorithm consists of two algorithms, each handling certain cases well. Each algorithm is run, and the better of the two results is chosen. This procedure will guarantee a better worst case bound than either algorithm alone. Since we will not know the exact ratio of C_A and C_E^*, we cannot tell in advance which of the two algorithms will guarantee a better worst case bound.

We introduce the notion of a polarity-constrained matching among a set of vertices which are heads and tails of arcs. A vertex which is a head is allowed to be matched only with a vertex which is a tail, and vice versa. The matching edges and the arcs taken together will thus form a number of cycles each of which can be traversed in a direction consistent with the arc directions. Thus, a stacker-crane tour is in effect a polarity- constained matching. A minimum-cost polarity-constrained matching is a polarity-constrained matching of least cost. Such a minimum-cost matching will often not be a tour, but rather a set of disjoint cycles.

We now present an algorithm which does well if the cost of the arcs is large relative to the cost of the edges. An example of Algorithm LARGEARC applied to a graph is shown in Figure 3.

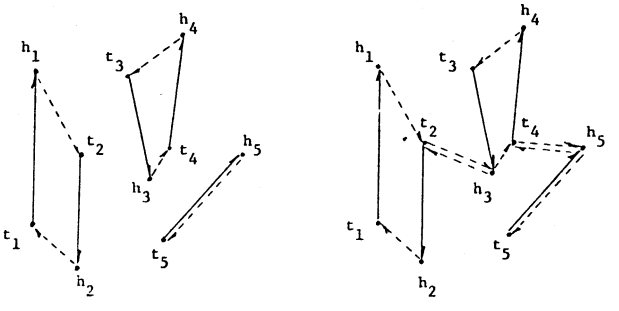

Figure 3 - LARGEARC Applied to a Graph

Algorithm LARGEARC:

Input: A mixed graph G' satisfying properties 1, 2, and 3.

Output: A sequence of edges and arcs representing a tour.

1. For each pair of arcs $\langle t_i, h_i \rangle$ and $\langle t_j, h_j \rangle$, set $c(h_i, h_j) = \infty$ and $c(t_i, t_j) = \infty$.

2. Perform a minimum cost matching on the vertices.

3. For each edge included in the matching, associate a direction with it, going from the vertex which is a head of an arc to the vertex which is a tail of an arc. (This results in $m \geq 1$ disjoint cycles consisting of alternating edges and arcs.)

4. Let the m disjoint cycles be R_i, $1 \leq i \leq m$, with each R_i represented by a single node n_i. Form the inter-node distances from the original edge costs:

$d(n_i,n_j) = \min_{u \in R_i, v \in R_j}\{c(u,v) | u \in R_i, v \in R_j\}$.
Associate with (n_i,n_j) the particular edge (u,v) which yields the minimum cost.

5. Find a minimum cost spanning tree for the $\{n_i | 1 \leq i \leq m\}$, using the distance function d.

6. Rename each spanning edge in terms of the original vertices. Make two copies of each edge, associating one direction with one edge, and the opposite direction with the other edge.

7. Call POSTPROCESS. []

We next present an algorithm that performs well if the cost of the arcs is small relative to the the cost of the edges. An example of Algorithm LARGEEDGE applied to a graph is shown in Figure 4.

Algorithm LARGEEDGE:
Input: Same as LARGEARC.
Output: Same as LARGEARC.

1. Let node n_i represent each arc $\langle t_i,h_i \rangle$. Set the inter-node distances:
$d(n_i,n_j) = \min\{c(t_i,t_j),c(t_i,h_j),c(h_i,t_j),c(h_i,h_j)\}$
Associate with the edge (n_i,n_j) the particular edge which yields the minimum cost.

2. Find a minimum cost spanning tree for the $\{n_i\}$, using the distance function d.

3. Identify nodes of odd degree in the spanning tree, and perform a minimum cost matching on these nodes using the distance function d.

4. Rename the spanning edges and matching edges in terms of the original vertices. Consider the degree of the vertices with spanning edges, matching edges, and arcs. For all arcs $\langle u,v \rangle$ whose endpoints have odd degree, add the edge (u,v) and associate with it the direction opposite that of the arc. Those arcs requiring no such edge shall be called even arcs, with total cost C'_A.

5. Traverse the graph ignoring arc directions. If the cost of the even arcs which are traversed backwards is more than $1/2\ C'_A$, then reverse the direction of the traversal.

6. Associate with each undirected edge the direction of the traversal. For each even arc that is incorrectly traversed, add two edges, both with associated direction opposite that of the arc.

7. Call POSTPROCESS. []

We complete the specification of the algorithm by giving a postprocessing algorithm and then the complete algorithm with subroutine calls. Step 1 in Algorithm POSTPROCESS can make use of an algorithm for traversing directed graphs where all vertices have the same indegree as outdegree, such as the algorithm in Edmonds and Johnson [4]. We complete the example of Figure 4 in Figure 5, showing the effect of Step 2 in POSTPROCESS.

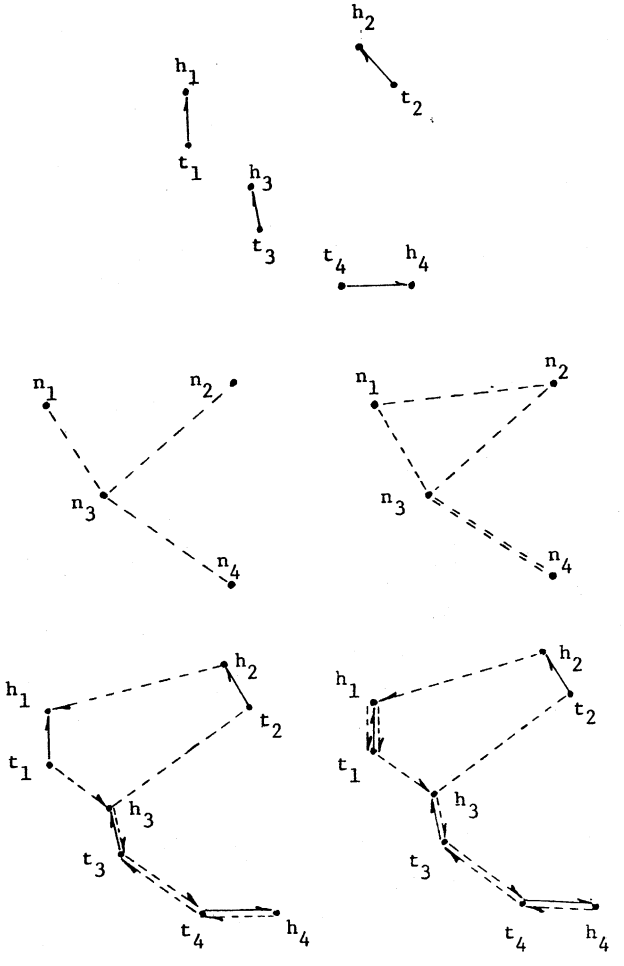

Figure 4 - LARGEEDGE Applied to a Graph

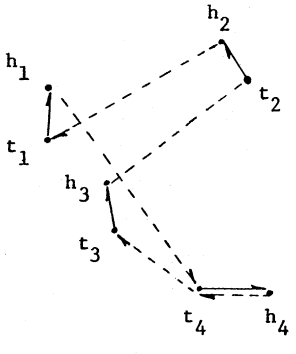

Figure 5 - Postprocessing a Tour

Algorithm POSTPROCESS:

Input: A set of edges and arcs from which a tour can be constructed.

Output: A sequence of edges and arcs representing a tour.

1. Given a set of arcs and edges with an associated direction, such that a traversal exists, find the traversal.

2. For any series of two or more consecutive edges in the tour, replace them with one edge, thus eliminating the intermediate vertices.

3. Translate all vertices to the names which they possessed before application of Algorithm PREPROCESS. (Undo Step 2 of PREPROCESS). For any two vertices v_i and v_j anchoring an edge in the tour, insert any vertices that would create a shorter path between v_i and v_j. (Undo Step 3 of PREPROCESS).

4. List the tour beginning at v_s, the initial vertex, and finishing at v_f, the terminal vertex. []

Algorithm CRANE:

Input: A mixed graph G satisfying property 2.

Output: A sequence of edges and arcs representing a tour.

1. Call PREPROCESS.
2. Call LARGEARC.
3. Call LARGEEDGE.
4. Select the tour of smaller cost. []

We now verify that the tours created by LARGEARC and LARGEEDGE are in fact traversable, and analyze the cost of the tours in the worst case.

Lemma 1. Algorithm LARGEARC produces a tour which is a sequence of arcs and edges, whose cost is at most $C_A + 3C_E^*$.

Proof. The polarity-constrained matching in steps 1, 2, and 3 produces a set of disjoint cycles, each of which can be traversed consistent with arc direction. The indegree and outdegree of each node are one. The spanning tree edges created in steps 4 and 5 connect the cycles. Since two of each spanning edge are added, with one edge having the opposite direction from the other, the indegree and outdegree of each vertex are still equal.

The cost of the matching, including the arcs, in steps 1 and 2 will be at most $C^* = C_A + C_E^*$. The optimal tour is in fact a polarity-constrained matching, and can be no smaller than the minimum polarity-constrained matching. The spanning tree edges in steps 4 and 5 must be smaller than C_E^*, since all arcs must be connected in the optimal tour. Step 6 doubles the spanning edges, so that the graph will have cost at most $C_A + 3C_E^*$. []

Lemma 2. Algorithm LARGEEDGE produces a tour which is a sequence of arcs and edges, whose cost is at most $2C_A + 3/2\ C_E^*$.

Proof. Step 1 has the effect of collapsing each arc to a single node. Steps 2 and 3 create a connected graph in which all nodes have even degree. A node actually represents two vertices, the head and the tail of an arc. If the degree of a node is even, then both vertices must either be of even degree or odd degree. If odd, then Step 4 adds an edge which makes both vertices even, so that after completion of Step 4, all vertices are of even degree. Steps 5 and 6 show how to augment the graph to make it traversable with respect to arc direction. Given a traversal which ignores arc direction, step 6 adds two edges for each incorrectly traversed arc. One edge will create a cycle with the arc, and the second edge can be traversed in a direction consistent with the tour. Since two edges are added, the vertices will remain of even degree.

Step 1 has the effect of creating a traveling salesman problem. Since the minimum inter-arc distances were selected, the cost of an optimal traveling salesman tour will be at most C_E^*. Steps 2 and 3 are Christofides' approximation algorithm for the traveling salesman problem, which guarantees a solution no worse than 3/2 times optimal. Thus the cost of the spanning edges in step 2 and the matching edges in step 3 is at most 3/2 times optimal. The cost of the edges added in step 4 is $C_A - C_A'$, since no arc is shorter than its corresponding edge. The cost of the extra edges added in step 6 is at most $2 * 1/2\ C_A'$. The cost of all edges added in steps 4 and 6 is at most $C_A - C_A' + 2 * 1/2\ C_A' = C_A$. The cost of the arcs, spanning edges, matching edges, and additional edges is $C_A + 3/2\ C_E^* + C_A$. []

Theorem 2. If C^* is the cost of an optimal tour for the stacker-crane problem, and $\hat{C}$ is the cost of the tour generated by Algorithm CRANE, then

$$\hat{C}/C^* \leq 9/5 .$$

The time complexity of Algorithm CRANE is $O(n^3)$, where n is the number of arcs.

Proof. If $C_A \geq 3/2\ C_E^*$, consider the results of Algorithm LARGEARC, from Lemma 1.

$$\hat{C}/C^* \leq (C_A + 3C_E^*)/(C_A + C_E^*)$$
$$\leq 1 + 2C_E^*/(C_A + C_E^*)$$
$$\leq 1 + 2C_E^*/(5/2\ C_E^*)$$
$$\leq 9/5 .$$

If $C_A < 3/2\ C_E^*$, consider the results of Algorithm LARGEEDGE, from Lemma 2.

$$\hat{C}/C^* \leq (2C_A + 3/2\ C_E^*)/(C_A + C_E^*)$$
$$\leq 2 - 1/2\ C_E^*/(C_A + C_E^*)$$
$$\leq 2 - 1/2\ C_E^*/(5/2\ C_E^*)$$
$$\leq 9/5 .$$

We now consider the time complexity of CRANE. Algorithm PREPROCESS is dominated by the shortest path algorithm, which can be done in $O(n^3)$ time. Algorithms LARGEARC and LARGEEDGE are both dominated by the minimum-cost matching algorithm, which Gabow and Lawler [5] have shown to be no worse than $O(n^3)$. []

3. k-Person Routing Problems

In this section we consider three different k-person routing problems. We first show that the recognition versions of all three problems are NP-complete. We then present and analyze heuristic algorithms which yield approximate solutions.

3.1 NP-Completeness of k-Person Problems

To show the NP-completeness of the three routing problems, we will need the following problem.

k-PP (k-Partition Problem, $k \geq 2$): Given a multiset $S = \{a_1,\ldots,a_m\}$, with $A = \sum_{i=1}^{m} a_i$ divisible by k. Decide if there exists a partition $S_1,\ldots,S_k$ such that $\sum_{a \in S_j} a = A/k$ for all $1 \leq j \leq k$.

We state a result from Sahni and Gonzalez [15] as a lemma.

Lemma 3. k-PP is NP-complete for $k \geq 2$.

Theorem 3. k-TSP is NP-complete for $k \geq 2$.

Proof. It is easy to see that k-TSP is in NP.

We show that we can transform k-PP into k-TSP in polynomial time. Let $S = \{a_1,\ldots,a_m\}$ be an instance of k-PP, Form a complete graph $G = (V,E)$, where $V = \{v_s,v_1,v_2,\ldots,v_m\}$. For each $v_i \neq v_s$, set $c(v_s,v_i) = a_i$. For all pairs $v_i \neq v_s$, $v_j \neq v_s$, set $c(v_i,v_j) = a_i + a_j$. Let $C = (2/k) \sum_{i=1}^{m} a_i$. (Refer to Figure 6.)

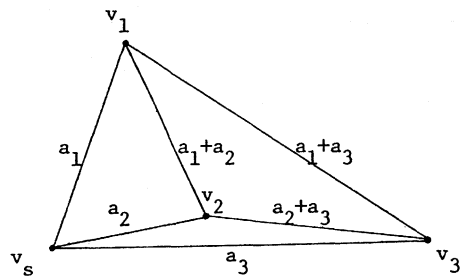

Figure 6 - Transforming k-PP into k-TSP

Suppose a solution to k-PP is $S_1,S_2,\ldots,S_k$, where $S_i = \{a_{i1},\ldots,a_{in_i}\}$. Then the image k-TSP problem has a solution of k tours R_i, $1 \leq i \leq k$, such that $R_i = (v_s,v_{i1},\ldots,v_{in_i},v_s)$ and is of cost C. Conversely, a solution of the image k-TSP problem can be translated into a solution for k-PP, $S_i = \{a_j \mid v_j$ is in R_i and $v_j \neq v_s\}$. []

Theorem 4. k-CPP is NP-complete for $k \geq 2$.

Proof. k-CPP is in NP.

We show that k-PP is reducible to k-CPP. Let $S = \{a_1,\ldots,a_m\}$ be an instance of k-PP. Form a multigraph with a single vertex v_s, and an edge of cost a_i for every element in S. Let $C = (1/k) \sum_{i=1}^{m} a_i$. (See Figure 7.)

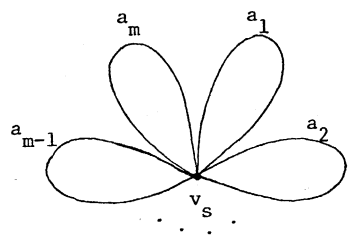

Figure 7 - Transforming k-PP into k-CPP

It is clear that k-PP has a solution if and only if the image k-CPP problem has a solution. []

Theorem 5. k-SCP is NP-complete for $k \geq 2$.

Proof. k-SCP is obviously in NP.

We reduce k-TSP to k-SCP. Perform the same transformation as in the proof of Theorem 1, when reducing TSP to SCP.

A k-tour for k-TSP in which no cycle has cost greater than C can be translated into a k-tour for k-SCP so that no cycle has cost greater than C. The converse is also true. []

3.2 Approximation Algorithms for k-TSP

We have considered two basically different methods for building a k-tour. The first method is to build k subtours simultaneously, modifying a heuristic of an incremental nature that generates an approximate solution for 1-person problems. The second method is to build a k-tour by splitting a good tour for 1 person into k subtours.

For the traveling salesman problem, there are several well known heuristic algorithms that find a tour by adding vertices one by one to a subtour. Among these are the nearest neighbor, nearest insertion, cheapest insertion, and farthest insertion algorithms analyzed by Rosenkrantz, et al., [13]. We shall consider generalizations of the nearest neighbor and nearest insertion algorithms to handle k salesmen. We shall find their algorithms to perform rather poorly in worst case.

Let $R = (v_{i_1},\ldots,v_{i_m})$ be a subtour, where $v_{i_1} = v_{i_m}$. For a vertex u not in R, we define the distance from u to R by $c(u,R) = \min_{v\in R}\{ c(u,v) | v \in R\}$. The cost of inserting a vertex u between v_{i_l} and $v_{i_{l+1}}$ is $c(v_{i_l},u) + c(u,v_{i_{l+1}}) - c(v_{i_l},v_{i_{l+1}})$, $1 \le l < m$. We say that a vertex u is inserted into a subtour R if u is inserted between two vertices in R with the minimum cost, and denote it by $R\leftarrow(u)$. The cost c(R) of a subtour R is the sum of costs of all edges in R. Let $R_1,\ldots,R_k$ be k subtours. We define their cost $c(R_1,\ldots,R_k)$ as the maximum of the costs of each subtour, i.e., $c(R_1,\ldots,R_k) = \max\{c(R_i)\}$.

Algorithm k-NEARINSERT:

1. Start with k subtours $R_j = (v_s,v_s)$, $1 \le j \le k$, where v_s is the start vertex.

2. For each j, $1 \le j \le k$, find a vertex u_j not currently in any subtour such that $c(u_j,R_j)$ is minimal.

3. Let i be such that $c(R_1,\ldots,R_i\leftarrow(u_i),\ldots,R_k)$ is minimum. Insert u_i into the subtour R_i.

4. If $(R_1,\ldots,R_k)$ covers all vertices, then stop. Otherwise go to step 2. []

Lemma 4. If $\hat{C}_k$ is the cost of the largest of the k subtours generated by algorithm k-NEARINSERT, and C_1^* is the cost of an optimal tour for one salesman, then

$$\hat{C}_k/C_1^* < 2 .$$

Proof. Let $(R_1,\ldots,R_k)$ be a k-tour with $R_i = (v_s,v_{i_1},\ldots,v_{i_{m_i}},v_s)$. Let $G_i = (V_i,E_i)$ be the complete subgraph of G, where $V_i = \{v_s,v_{i_1},\ldots,v_{i_{m_i}}\}$. Suppose the subtour R_i has been built by adding vertices $v_{p(i_1)},v_{p(i_2)},\ldots,v_{p(i_{m_i})}$ in this order, where p is a permutation of $(i_1,i_2,\ldots,i_m)$. Then R_i is a tour of G_i that is obtained by the 1-person nearest insertion algorithm on $G_i = (V_i,E_i)$. Thus if F_i^* is the cost of an optimal tour of G_i, then $c(R_i)/F_i^* < 2$ [13, Thm. 3]. Since $\hat{C}_k = \max\{c(R_i)\}$, and $F_i^* \le C_1^*$ for all i, $1 \le i \le k$, then $\hat{C}_k/C_1^* < 2$. []

Theorem 6. If C_k is the cost of the largest of the k subtours generated by algorithm k-NEARINSERT, and C_k^* is the cost of the largest subtour in an optimal solution of k-TSP, then

$$\hat{C}_k/C_k^* < 2k ,$$

and the bound is approachable.

Proof. By Lemma 4, $\hat{C}_k < 2C_1^*$. By the triangle inequality $C_1^* \le k C_k^*$. Thus $\hat{C}_k/C_k^* < 2k$. To show that the bound is approachable, we consider the following example. Let G_n be a graph of n vertices shown in Figure 8, which is taken from [13, Thm. 14]. Let the cost of all edges not shown be 2. Let $R_n' = (v_1,v_3,\ldots,v_n,v_{n-1},v_{n-3},\ldots,v_1)$ be a tour for the single TSP obtained by 1-NEARINSERT. Then $c(R_n') = 2(n-1)$ and $C_1^* = n$.

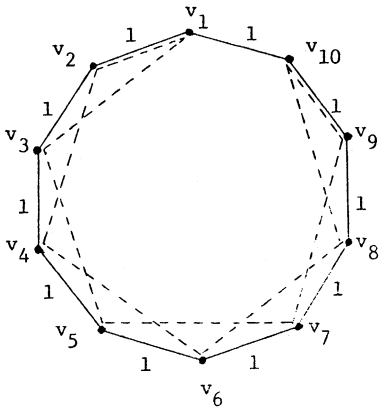

Figure 8 - G_{10} and a 1-tour by 1-NEARINSERT

Consider the graph G_n^k which is k copies of G_{kn} with v_1 in common. We denote the vertices of the jth copy $G_{j,kn}$ as $v_1 = v_{j,1},v_{j,2},\ldots,v_{j,kn}$. The cost of edges in $G_{j,kn}$ is the same as in G_{kn}. The cost of edges between vertices in different copies, $v_{j,l}$ and $v_{i,m}$ is 0 if $l = m$; is 1 if $|l - m| = 1$, or if $l = 1$ and $m = kn$; and is 2 otherwise. A k-tour by k-NEARINSERT is $(R_1,\ldots,R_k)$, where for each $1 \le j \le k$, $R_j = (v_{j,1},v_{j,3},\ldots,v_{j,kn},v_{j,kn-1},v_{j,kn-3},\ldots,v_{j,1})$ with $c(R_j) = 2(kn - 1)$. Thus $\hat{C}_k = 2(kn - 1)$. On

the other hand, an optimal k-tour is $(R_1^*,\ldots,R_k^*)$, where

$$R_1^* = (v_1,v_{1,2},v_{2,2},\ldots,v_{k,2},v_{1,3},\ldots,v_{k,3},\ldots,v_{i,n},\ldots,v_{k,n},v_1),\ldots,$$
$$R_k^* = (v_1,v_1,(k-1)n+1,\ldots,v_k,(k-1)n+1,\ldots,v_{1,kn},\ldots,v_{k,kn},v_1).$$

Thus $c(R_j^*) = n + 1$, and $C_k^* = n + 1$. Hence

$$\hat{C}_k/C_k^* = 2(kn-1)/(n+1) = 2k - 2(k+1)/(n+1)$$

which approaches 2k for large n. []

The following notation is used in describing the algorithm k-NEARNEIGHBOR. Let $R = (v_{i_1},\ldots,v_{i_m})$ be a path. We call v_{i_1} the initial vertex and v_{i_m} the terminal vertex of R. A vertex u is added to R by connecting u to the terminal vertex, and is denoted by R<--u.

Algorithm k-NEARNEIGHBOR:

1. Set each of the k paths initially to the initial vertex, i.e. $R_j = (v_s)$ for all j, $1 \le j \le k$.

2. For each j, $1 \le j \le k$, let u_j be the vertex nearest to the terminal vertex of R_j.

3. Let i be such that $c(R_1,\ldots,R_i<--u_i,\ldots,R_k)$ is minimum. Add u_i to R_i.

4. If there are vertices remaining to be added to a path then go to step 2. Otherwise build a k-tour from the R_j's by connecting their terminal vertices to their initial vertices. []

Lemma 5. If $\hat{C}_k$ is the cost of the largest of the k subtours obtained by k-NEARNEIGHBOR, and C_1^* is the cost of an optimal tour for one salesman, then

$$\hat{C}_k/C_1^* < (1/2)\log n + 1 .$$

Proof. A proof similar to that for Lemma 4 using the result in [13, Thm. 1] applies. []

Theorem 7. If $\hat{C}_k$ is as in Lemma 5, and C_k^* is the cost of the largest subtour in an optimal solution of k-TSP, then

$$\hat{C}_k/C_k^* < (k/2)\log n + k .$$

Also, there is a graph for which $\hat{C}_k/C_k^* > (k/6)\log n$.

Proof. By the triangle inequality, $C_1^* \le k\, C_k^*$, and by Lemma 5, $\hat{C}_k < (1/2)\log n + 1$. Thus $\hat{C}_k/C_k^* < (k/2)\log n + k$. The second assertion can be proved as follows: Consider the complete graph $\bar{G}_{m-1} = (V,E)$ in [13, Thm. 2], where $|V| = n = 2^m - 1$. Use an argument similar to that in Theorem 6 to show that $\hat{C}_k > (n/3)\log n$ and $C_k^* < 2n/k$. []

We now describe an algorithm which employs a very simple tour-splitting heuristic. Given a tour for one traveling salesman, the algorithm splits the tour into k subtours of more or less equal cost. Obviously, the worst case behavior depends on the (generally) non-optimal tour which is split into k subtours. When a tour obtained by Christofides' algorithm is used, this simple heuristic is far superior to the heuristics already analyzed, in terms of worst case behavior. In the algorithm, c_{max} denotes $\max_{1\le i\le n} c(v_1,v_i)$, and for any path R, t(R) denotes its terminal vertex. An example of splitting a 1-tour into a 3-tour is shown in Figure 9.

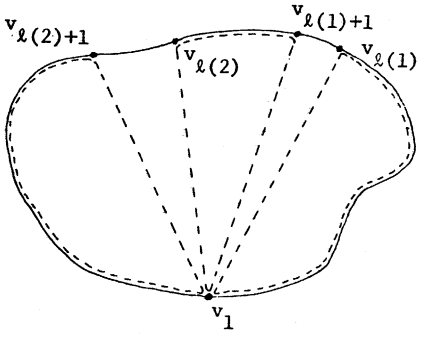

Figure 9 - Splitting a 1-tour into a 3-tour

Algorithm k-SPLITOUR:

1. Find a 1-tour $R = (v_1,v_2,\ldots,v_n,v_1)$ with $c(R) = L$, where v_1 is the initial vertex.

2. For each j, $1 \le j < k$, find the last vertex $v_{1(j)}$ such that the cost of the path from v_1 to $v_{1(j)}$ along R is not greater than $j/k\,(L - 2\,c_{max}) + c_{max}$.

3. Obtain the k-tour by forming k subtours as

$$R_1 = (v_1,\ldots,v_{1(1)},v_1),$$
$$R_2 = (v_1,v_{1(1)+1},\ldots,v_{1(2)},v_1),\ \ldots,$$
$$R_k = (v_1,v_{1(k-1)+1},\ldots,v_n,v_1).$$ []

Theorem 8. If $\hat{C}_k$ is the cost of the largest of the k subtours generated by Algorithm k-SPLITOUR, and C_k^* is the cost of the largest subtour in an optimal solution of k-TSP, then

$$\hat{C}_k/C_k^* \le e + 1 - 1/k$$

where e is the bound for the single traveling salesman algorithm.

Proof. The costs of the paths from v_1 to $v_{1(1)}$ and from $v_{1(k-1)+1}$ to v_1 along R are each no greater than $1/k\,(L - 2\,c_{max}) + c_{max}$. For each j, $1 \le j \le k-2$, the cost of the path from $v_{1(j)+1}$ to $v_{1(j+1)}$ is no greater than $1/k\,(L - 2\,c_{max})$. Thus for each j, $1 \le j \le k$, the cost of the tour R_j does not exceed $1/k\,(L - 2\,c_{max}) + 2\,c_{max}$. Therefore,

$$\hat{C}_k = \max_j c(R_j) \le 1/k\,(L - 2\,c_{max}) + 2\,c_{max} \le L/k + 2(1 - 1/k)c_{max}.$$

Due to the triangle inequality, $C_k^* \ge 1/k\ C_1^*$ and $c_{max} \le 1/2\ C_k^*$. Using these with $L \le e\ C_1^*$, we obtain $\hat{C}_k \le 1/k\ ekC_k^* + 2(1-1/k)C_k^*$, which yields $\hat{C}_k/C_k^* \le e + 1 - 1/k$. []

Corollary 3. There is an $O(n^3)$ approximation algorithm for k-TSP with bound

$$\hat{C}_k/C_k^* \le 5/2 - 1/k\ .$$

Proof. In step 1, we find a 1-tour R using Christofides' algorithm [2]. Since R can be found in $O(n^3)$ time, and $c(R)/C_1^* \le 3/2 = e$, the result is immediate. []

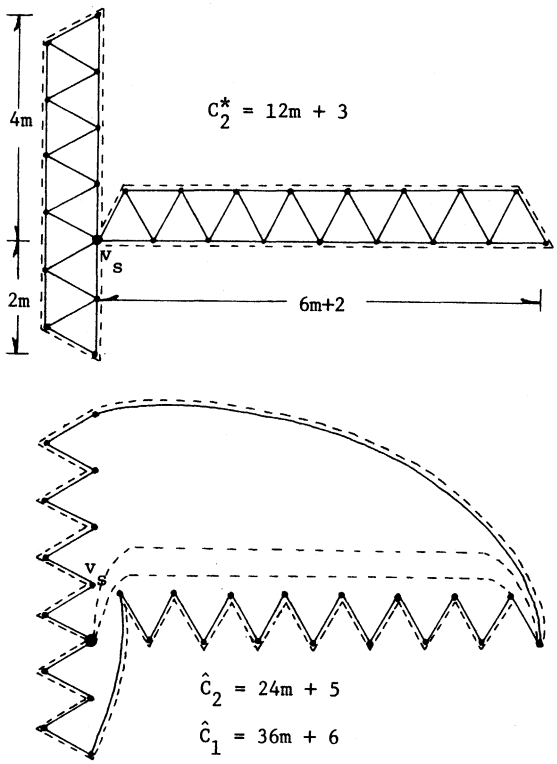

Figure 10 - Worst Case for 2-TSP

The bound in the corollary is tight if the algorithm in the proof is employed. An example for k = 2 which realizes the worst case bound of 5/2 - 1/2 = 2 is shown in Figure 10. All edges shown in Figure 10 (a) have unit cost, while all other edge costs are determined by the shortest path along the edges shown. An optimum 2-tour is shown in dotted lines in Figure 10 (a). Figure 10 (b) shows a 1-tour that could be produced by Christofides' algorithm in solid lines, and a 2-tour that could result from our algorithm in dotted lines.

3.3 Approximation Algorithms for k-CPP and k-SCP

The tour splitting heuristic yields a good bound for the Chinese Postman problem, since an optimal 1-route for the Chinese Postman problem can be obtained in polynomial time by using the algorithm of Edmonds and Johnson [4]. Let $R = (v_1,e_{i1},v_{i2},e_{i2},\ldots,v_{im},e_{im},v_1)$ be the 1-route so obtained. Let $L = c(R)$ and let $R_{v_{in}}$ denote the path $(v_1,e_{i1},v_{i2},\ldots,v_{in})$, with $n \le m$. We denote the cost of a shortest path from a vertex v to u by s(v,u). Define s_{max} as $1/2 \max_{2\le j\le m-1}\{s(v_1,v_{ij})+c(v_{ij}+v_{ij+1}) +s(v_{ij+1},v_1)\}$ and for each j, $1 \le j < k$, $L_j = j/k\ (L - 2\ s_{max}) + s_{max}$. Figure 11 shows the building of a k-route, and Figure 12 shows a completed 4-route.

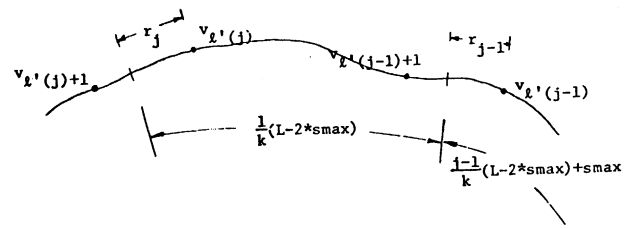

Figure 11 - Building a k-route

Algorithm k-POSTMEN:

1. Find an optimal 1-route $R = (v_1,e_{i1},v_{i2},\ldots, v_{im},e_{im},v_1)$ using the algorithm in [4], where v_1 is the start vertex.

2. For each j, $1 \le j \le k$, find the last vertex $v_{l'(j)}$ such that $c(R_{v_{l'(j)}}) \le L_j$.

3. Let $r_j = L_j - c(R_{v_{l'(j)}})$. For each j, $1 \le j < k$, if $r_j + s(v_{l'(j)},v_1) \le c(v_{l'(j)},v_{l'(j)+1}) - r_j + s(v_{l'(j)+1},v_1)$, then $v_{l(j)} = v_{l'(j)}$. Otherwise $v_{l(j)} = v_{l'(j)+1}$.

4. Let $R_1 = (v_1,e_{i1},v_{i2}, \ldots, v_{l(1)})$, $R_2 = (v_{l(1)},\ldots,v_{l(2)})$, $\ldots$, $R_k = (v_{l(k-1)},\ldots,v_1)$. Build the k-route by connecting v_1 to both the initial and terminal vertices of the R_j's with shortest paths to transform R_j into a subroute. []

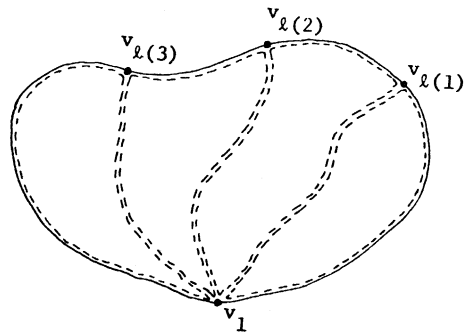

Figure 12 - A 4-route

We note that k-POSTMEN is more complicated than k-SPLITOUR. If the split point falls somewhere in the middle of an edge, then the algorithm must decide which subtour to include the edge in.

Theorem 9. The algorithm k-POSTMEN produces $\hat{C}_k$ in $O(n^3)$ time such that

$$\hat{C}_k/C_k^* \leq 2 - 1/k .$$

Proof. We consider the jth subroute R_j, $1 \leq j \leq k$. Since each edge in the graph must be traversed, $s_{max} \leq 1/2\ C_k^*$. By the definition of s_{max},

$$s(v_1,v_{1'(j)})+c(v_{1'(j)},v_{1'(j)+1}) +s(v_{1'(j)+1},v_1) \leq s_{max}$$

Hence

$$\min\{s(v_1,v_{1'(j)})+r_j,c(v_{1'(j)},v_{1'(j)+1})-r_j +s(v_{1'(j)+1},v_1)\} \leq s_{max}.$$

Similarly

$$\min\{s(v_1,v_{1'(j-1)})+r_{j-1},c(v_{1'(j-1)},v_{1'(j-1)+1}) -r_{j-1}+s(v_{1'(j-1)+1},v_1)\} \leq s_{max} .$$

The worst case for the jth subroute R_j is when it starts from v_1, reaches $v_{1'(j-1)}$, continues to $v_{1'(j)+1}$ along R and finally back to v_1. But in this case $s(v_1,v_{1'(j-1)})+r_{j-1} \leq s_{max}$ and

$$c(v_{1'(j)},v_{1'(j)+1})-r_j+s(v_{1'(j)+1},v_1) \leq s_{max}.$$

Thus $c(R_j) \leq s_{max} + 1/k\ (L - 2\ s_{max}) + s_{max}$.

Since $L \leq kC_k^*$ and $s_{max} \leq 1/2\ C_k^*$,

$$c(R_j) \leq 2\ s_{max}(1 - 1/k) + C_k^* \leq C_k^*(1 - 1/k) + C_k^* = C_k^*(2 - 1/k).$$

Other cases for R_j can easily be shown to satisfy the above inequality. Hence, we obtain $\hat{C}_k/C_k^* \leq 2 - 1/k$. []

Finally, we present a tour splitting algorithm for the k-stacker-cranes problem. The algorithm uses the same methods as k-SPLITOUR and k-POSTMEN with differences in detail. If a split point falls on an edge, a similar procedure is used as in k-SPLITOUR. If the split point falls on an arc, then a similar procedure as in k-POSTMEN is used. $R = (v_1,v_{i2},\ldots,v_{im},v_1)$, $R_{v_{in}}$, L, c_{max} and $L_j = j/k\ (L - 2\ c_{max}) + c_{max}$ are as previously defined.

Algorithm k-CRANES:

1. Find a 1-tour $R = (v_1,v_{i2},\ldots,v_{im},v_1)$ for the single crane problem with $c(R) = L$, where v_1 is the initial vertex.

2. For each j, $1 \leq j < k$, find the last vertex $v_{1'(j)}$ such that $c(R_{v_{1'(j)}}) \leq L_j$.

3. Let $r_j = L_j - c(R_{v_{1'(j)}})$. For each j, $1 \leq j < k$, if $(v_{1'(j)},v_{1'(j)+1})$ is an edge, then $v_{1(j)} = v_{1'(j)}$, and the terminal vertex of R_j is $v_{1'(j)}$ and the initial vertex of R_{j+1} is $v_{1(j)+1}$. Suppose $(v_{1'(j)},v_{1'(j)+1})$ is an arc. If

$$c(v_1,v_{1'(j)})+r_j \leq c(v_{1'(j)},v_{1'(j)+1})-r_j+c(v_{1'(j)+1},v_1),$$

then $v_{1(j)} = v_{1'(j)}$ and the initial vertex of R_{j+1} is $v_{1(j)}$. Also if $(v_{1(j)-1},v_{1(j)})$ is an arc, then the terminal vertex of R_j is $v_{1(j)}$, and if it is an edge then the terminal vertex of R_j is $v_{1(j)-1}$. If

$$c(v_1,v_{1'(j)})+r_j > c(v_{1'(j)},v_{1'(j)+1})-r_j+c(v_{1'(j)+1},v_1),$$

then $v_{1(j)} = v_{1'(j)+1}$ and the terminal vertex of R_j is $v_{1(j)}$. Also, if $(v_{1(j)},v_{1(j)+1})$ is an arc, then the initial vertex of R_{j+1} is $v_{1(j)}$ and otherwise the initial vertex of R_{j+1} is $v_{1(j)+1}$.

4. For each j, $1 \leq j \leq k$, construct the jth subtour R_j by connecting v_1 to the initial vertex of R_j and the terminal vertex of R_j to v_1 with direct edges. []

Theorem 10. Algorithm k-CRANES produces $\hat{C}_k$ such that

$$\hat{C}_k/C_k^* \leq e + 1 - 1/k$$

where e is the bound for a 1-crane algorithm.

Proof. A proof which is a combination of the proofs of Theorems 7 and 8 applies. []

Corollary 4. There is an approximation algorithm of $O(n^3)$ time complexity for k-SCP such that

$$\hat{C}_k/C_k^* \leq 14/5 - 1/k .$$

Proof. In step 1, a 1-tour may be obtained by using the algorithm in section 2, for which $e = 9/5$. []

4. Conclusion

In attacking the stacker-crane problem, we have applied a mixed strategy approach, which works well even though we cannot easily decide which of the two strategies will guarantee a better bound. We have introduced the notion of a polarity-constrained matching, which is useful in handling routing problems with directed edges. For the k-person routing problems, we have presented several heuristic algorithms with reasonable worst-case bounds.

Our work leaves several problems open. We have not been able to find an example which approaches the worst case bound for the stacker-crane problem. For the k-person problems, our tour-splitting heuristic is quite simple, and other heuristics may yield better results. It may also be fruitful to examine certain restricted cases such as a planar k-CPP or Euclidean k-CPP, k-TSP and k-SCP.

References

1. Bellmore,M. and Hong,S., "Transformation of the Multisalesmen Problem to the Standard Traveling Salesman Problem," JACM 21, 3, (July 1974), pp. 500-504.

2. Christofides,N., "Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem", to appear in Algorithms and Complexity: Recent Results and New Directions (1976).

3. Cook,S.A., "The Complexity of Theorem Proving Procedures," 3rd STOC (1971), 151-158.

4. Edmonds,J. and Johnson,E.L., "Matching, Euler Tours and the Chinese Postman,", Mathematical Programming 5 (1973) 88-124.

5. Gabow,H. and Lawler,E.L., "An Efficient Implementation of Edmonds' Algorithm for Maximum Weight Matching on Graphs," TR CU-CS-075-75, Dept. of Computer Science, Univ. of Colorado (1975).

6. Garey,M.R. and Johnson,D.S., "Approximation Algorithms for Combinatorial Problems: an Annotated Bibliography", to appear in Algorithms and Complexity: Recent Results and New Directions (1976).

7. Ibarra,O.H. and Kim,C.E., "Fast Approximation Algorithms for the Knapsack and Sum of Subset Problems," JACM 22, 4 (1975) 463-468.

8. Johnson,D.S., "Approximation Algorithms for Combinatorial Problems," J Comput Systems Sci 9 (1974) 256-278.

9. Karp,R.M., "Reducibility Among Combinatorial Problems," in Complexity of Computer Computations, R.E.Miller and J.W.Thatcher, Eds., Plenum Press, N.Y. (1972) 85-104.

10. Miller,C.E., Tucker,A.W., and Zemlin,R.A., "Integer Programming Formulation of Traveling Salesman Problem," JACM 7, (1960), pp. 362-329.

11. Papadimitriou,C.H. and Stieglitz,K., "Some Complexity Results for the Traveling Salesman Problem," 8th STOC (1976), pp. 1-9.

12. Rosenkrantz,D.J., private communication (1976).

13. Rosenkrantz,D.J., Stearns,R.E., and Lewis,P.M., "Approximate Algorithms for the Traveling Salesperson Problem," 15th SWAT (1974).

14. Sahni,S.K., "Approximate Algorithms for the 0/1 Knapsack Problem," JACM 22 (1975) 115-124.

15. Sahni,S.K., and Gonzalez,T., "P-Complete Approximation Problems," TR 75-12, Dept. of Computer Science, Univ. of Minnesota (1975).