SIAM J. COMPUT.
Vol. 7, No. 2, May 1978

APPROXIMATION ALGORITHMS FOR SOME ROUTING PROBLEMS*

GREG N. FREDERICKSON†, MATTHEW S. HECHT‡ AND CHUL E. KIM¶

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.

Key words. *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. Routing problems that involve the periodic collection and delivery of goods and services are of great practical importance. Common examples of such problems include mail delivery, newspaper delivery, parcel pickup and delivery, trash collection, schoolbus routing, snow removal, and fuel oil delivery. The practical goals of scrutinizing such problems are cost minimization and service improvement. Abstractions of these problems can be modeled easily and naturally with graphs.

Unfortunately, many of the interesting routing problems, including for instance the well known traveling salesman problem, are *NP*-complete in the sense of Cook [3] and Karp [10]. The problems we consider in this paper, the stacker-crane problem and three k-person routing problems, are also *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, attention has been given to developing algorithms that solve various *NP*-complete problems efficiently but only approximately [7], [8], [9], [14]. We shall restrict our attention to approximation algorithms for routing problems for which worst-case analysis has been performed.

Previous worst-case analysis of approximation algorithms for routing problems has focused on the traveling salesman problem. Sahni and Gonzalez [16] have shown that if the triangle inequality is not satisfied, the problem of finding an approximate solution for the traveling salesman problem within any constant bound ratio of the optimum is as difficult as finding an exact solution. Papadimitriou and Steiglitz [12] 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 [14] have applied worst-case analysis to several incremental (insertion) heuristics. They show that none of the algorithms examined have a worst-case bound better than 2. This bound is the same as 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

* Received by the editors July 23, 1976, and in final revised form June 27, 1977.

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

‡ Department of Computer Science, University of Maryland, College Park, Maryland 20742. This research was partially supported by the National Science Foundation under Grant DCR-08361.

¶ Department of Computer Science, University of Maryland, College Park, Maryland 20742. This research was partially supported by a University of Maryland General Research Board Award.

odd degree. We make use of Christofides' techniques in achieving a worst-case bound of 1.8 for an algorithm for the stacker-crane problem.

The stacker-crane problem, which was brought to our attention by Rosenkrantz [13], is a modified traveling salesman problem that 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. The goal is 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.

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 $k \geqq 2$. 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 [11].

We have found no worst-case analysis for any k-person routing problems in the literature. 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-TSP, k-CPP, and k-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* $\langle t, h\rangle$ is a directed edge from t to h, both vertices in V. We call t the *tail* of arc $\langle t, h\rangle$, and h the *head* of arc $\langle t, h\rangle$.

A multiset is a set with a function mapping the elements of the set into the positive integers, to indicate that an element may appear more than once. A *multigraph* is a graph $G=(V, E)$ 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 mixed graph $G=(V, E, A)$, a *path* is a sequence of vertices such that for each adjacent pair of vertices v_i and v_{i+1}, there is either an edge (v_i, v_{i+1}) or an arc $\langle v_i, v_{i+1}\rangle$ in the graph. 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 visiting a subset of the vertices in V. 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, such that each edge connects the immediately preceding and

succeeding vertices. A *subroute* is a route which covers a subset of E. An arc $\langle t, h\rangle$ is *traversed* by covering it in a direction from its tail t to its head h.

A (maximum cardinality) *matching* $E' \subseteq E$ of a graph $G = (V, E)$ is a (maximal) subset of edges which share no vertices. A *minimum-cost matching* of G is a matching such that the total cost of the edges between the paired vertices is minimum. Given a partition $V = V_1 \cup V_2$, a *bipartite matching* is a matching of $G' = (V, (V_1 \times V_2) \cap E)$. 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 define the stacker-crane problem (SCP) as follows: 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, such that for every arc there is a parallel edge of no greater cost. 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.

As stated previously, the stacker-crane problem is *NP*-complete. An instance of TSP can be transformed 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 Fig. 1 for an example of the translation.) Choose any vertex v_{it} and designate it the initial vertex.

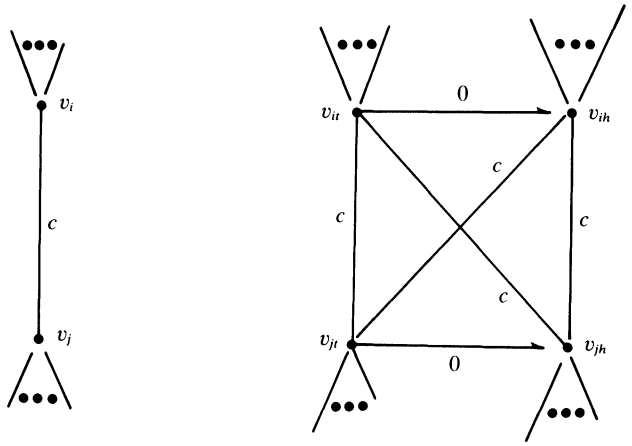

FIG. 1. *Translation from* TSP *to* SCP.

A tour of cost at most C for TSP can be translated into a tour of cost at most C for SCP. 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 SCP can be translated into a tour of cost at most C for TSP by changing v_{ih} to v_i and deleting v_{it}, for all v_i in V.

We now consider approximation algorithms 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 at least one arc in A.
2. The cost function on edges satisfies the triangle inequality.

If a graph does not satisfy the preceding properties, Algorithm PREPROCESS will transform the graph into an equivalent graph which satisfies these properties.

ALGORITHM PREPROCESS.

Input: A mixed graph $G = (V, E, A)$ and a cost function c.

Output: A mixed graph $G' = (V', E', A)$ satisfying properties 1 and 2, and a cost function c'.

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. Insert all vertices which are endpoints of arcs into V'. Calculate the shortest path between every pair of vertices v_i and v_j in V', insert (v_i, v_j) into E', and set $c(v_i, v_j)$ to the cost obtained.

We shall use the following notation in the analysis of the worst-case behavior of our algorithm. Let C^* represent the cost of an optimal tour, and let C_A be the total cost of all arcs.

We consider two strategies which depend on the relative size of C_A as compared with C^*. If C_A is large relative to C^*, then the analogue of a minimum spanning tree for the arcs will be small. An appropriate strategy in this case is to perform a minimum cost matching on the heads and tails of arcs, and link the resulting cycles together with the spanning edges. If C_A is small relative to C^*, then the problem is essentially a traveling salesman problem. In this case shrink the arcs to nodes, perform Christofides' algorithm [2], and add certain edges corresponding to the arcs to make the tour traverse arcs correctly.

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 know of no efficient way to compute the exact ratio of C_A to C^*, it is difficult to know in advance which of the two algorithms will guarantee a better worst-case bound.

We now present an algorithm which does well if the cost of the arcs is large relative to the cost of the optimum tour. An example of Algorithm LARGEARCS applied to a graph is shown in Fig. 2.

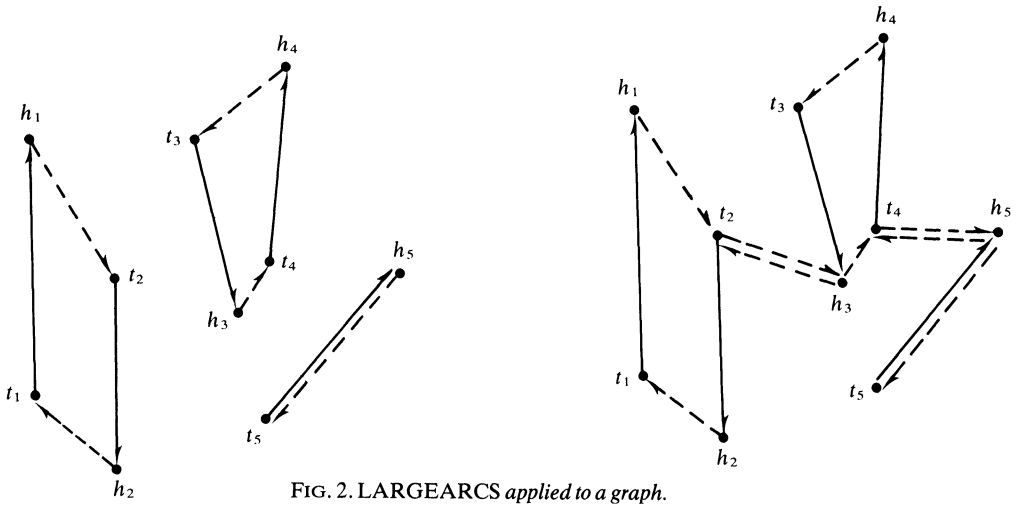

FIG. 2. LARGEARCS *applied to a graph.*

ALGORITHM LARGEARCS.

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

Output: A sequence of vertices representing a tour.

1. Find a minimum cost bipartite matching between the multisets of heads and tails of arcs.
2. Initialize E'' to be empty. 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, and insert it into E''. (This results in $m \geqq 1$ disjoint cycles consisting of alternating edges and arcs.)
3. Let the m disjoint cycles be R_i, $1 \leqq i \leqq m$, with each R_i represented by a single node n_i. Form the inter-node distances from the original edge costs:

$$d(n_i, n_j) = \min \{c'(u, v) | u \in R_i, v \in R_j\}.$$

   Associate with (n_i, n_j) the particular edge (u, v) which yields the minimum cost.
4. Find a minimum cost spanning tree for the nodes $\{n_i \mid 1 \leqq i \leqq m\}$, using the distance function d.
5. 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, and insert these edges into E''.
6. Call POSTPROCESS.

We next present an algorithm that performs well if the cost of the arcs is small relative to the cost of the optimum tour. An example of Algorithm SMALLARCS applied to a graph is shown in Fig. 3.

ALGORITHM SMALLARCS.

Input: Same as LARGEARCS.

Output: Same as LARGEARCS.

1. Represent each arc $\langle t_i, h_i \rangle$ by a node n_i. For each pair of nodes n_i and n_j, define $c'(n_i, n_j)$ as the minimum of $c(t_i, t_j)$, $c(t_i, h_j)$, $c(h_i, t_j)$ and $c(h_i, h_j)$. Perform an all shortest paths algorithm using c' to find the inter-node distance $d(n_i, n_j)$. Associate with each edge (n_i, n_j) the edges in the shortest path between n_i and n_j.
2. Find a minimum cost spanning tree for the nodes $\{n_i\}$, 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 vertices in V'. Define $G'' = (V, E'', A)$, where E'' is the multiset of spanning edges and matching edges. Consider the degree of vertices in G''. For all arcs $\langle u, v \rangle$ whose endpoints have odd degree, add the edge (u, v) to E'' and associate with it the direction opposite that of the arc. These arcs requiring no such edge shall be called *even arcs*, with total cost C'_A.
5. Find a tour which exactly covers $E'' \cup A$, ignoring even 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 tour.
6. Associate with each undirected edge the direction of the tour. For even arc that is incorrectly traversed, add two edges to E'', both with associated direction opposite that of the arc.
7. Call POSTPROCESS.

We next present a postprocessing algorithm, and then finally the complete algorithm with subroutine calls. Step 1 in Algorithm POSTPROCESS can make use of

an algorithm for finding tours in 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 Fig. 3 in Fig. 4, showing the effect of step 2 in POSTPROCESS.

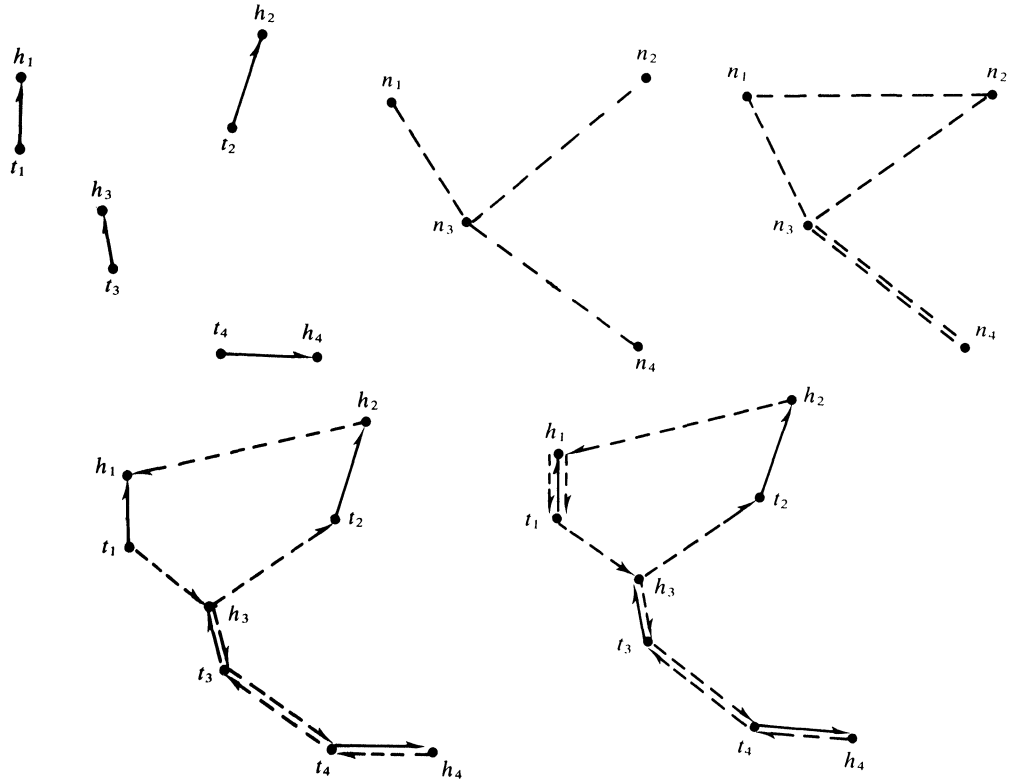

FIG. 3. SMALLARCS *applied to a graph.*

ALGORITHM POSTPROCESS.
Input: A set of edges and arcs from which a tour can be constructed.
Output: A sequence of vertices representing a tour.

1. Given a set of arcs and edges with an associated direction, from which a tour can be constructed, find the tour.
2. For any series of two or more consecutive edges in the tour, replace them with one edge, thus eliminating the intermediate vertices.
3. 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 2 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.
Output: A sequence of vertices representing a tour.

1. Call PREPROCESS.
2. Call LARGEARCS.
3. Call SMALLARCS.
4. Select the tour of smaller cost.

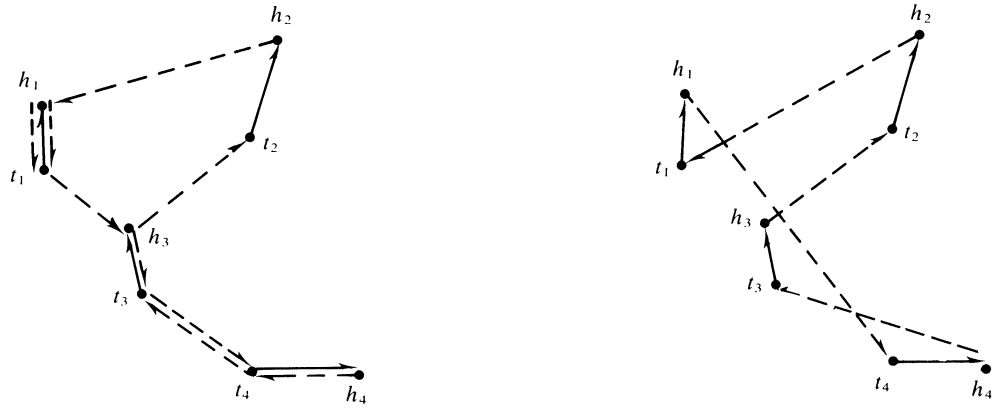

FIG. 4. *Postprocessing a tour.*

We now verify that the tours created by LARGEARCS and SMALLARCS do in fact traverse all of the arcs, and we analyze the cost of the tours in the worst case.

LEMMA 1. *Algorithm* LARGEARCS *produces a tour which traverses all of the arcs, and whose cost is at most* $3C^* - 2C_A$.

Proof. The bipartite matching of heads and tails of arcs in step 1 produces a set of edge disjoint cycles, each of which is consistent with arc directions. The indegree and outdegree of each nodes are thus equal. The spanning tree edges created in steps 3 and 4 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 in G'' are still equal. Thus step 6 will produce a tour from $E'' \cup A$.

The cost of the matching, including the arcs, in steps 1 and 2 will be at most C^*. An optimum tour is in fact a bipartite matching of heads and tails of arcs, and can be no smaller than the minimum cost bipartite matching. The cost of the spanning tree edges in steps 3 and 4 must be smaller than $C^* - C_A$, since all arcs must be included in the optimum tour. Step 5 doubles the spanning edges, so that the tour produced will have cost at most $3C^* - 2C_A$. □

LEMMA 2. *Algorithm* SMALLARCS *produces a tour which traverses all of the arcs, and whose cost is at most* $(3/2)C^* + (1/2)C_A$.

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 allow the tour to traverse arcs in the proper direction. Given a tour 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. Thus step 7 can produce a tour from $E'' \cup A$.

Step 1 has the effect of creating a traveling salesman problem. Since the minimum inter-arc distances were selected, the cost of an optimum traveling salesman tour will be at most $C^* - C_A$. 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 optimum. 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 $C_A + (3/2)(C^* - C_A) + C_A$. □

THEOREM 1. *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*

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

The time complexity of Algorithm CRANE *is* $O(\max\{|V|^3, |A|^3\})$.

Proof. If $C_A \geqq (3/5)C^*$, consider the results of Algorithm LARGEARCS, from Lemma 1.

$$\hat{C}/C^* \leqq (3C^* - 2C_A)/C^*$$
$$\leqq (3C^* - (6/5)C^*)/C^* = 9/5.$$

If $C_A < (3/5)C^*$, consider the results of Algorithm SMALLARCS, from Lemma 2.

$$\hat{C}/C^* \leqq ((3/2)C^* + (1/2)C_A)/C^*$$
$$\leqq ((3/2)C^* + (3/10)C^*)/C^* = 9/5.$$

We now consider the time complexity of CRANE. Algorithm PREPROCESS is dominated by the all shortest paths algorithm [5], which requires $O(|V|^3)$ time. Algorithm LARGEARCS is dominated by the algorithm for weighted bipartite matching. The weighted general matching algorithm of Edmonds and Johnson [4] as implemented by Gabow and Lawler [6] can be used and will require $O(|A|^3)$ time. Algorithm SMALLARCS is dominated by the all shortest paths algorithm, which requires $O(|A|^3)$ time, and the weighted general matching algorithm, which is no worse than $O(|V|^3)$. □

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.

k-TSP (k-traveling salesmen problem, $k > 1$): 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, $k > 1$): 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, $k > 1$): 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 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 $k > 1$, 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.

For $k > 1$, k-TSP, k-CPP, and k-SCP are all *NP*-complete. The k-partition problem, which Sahni and Gonzalez [16] have shown to be *NP*-complete, can be reduced to the k-person problems:

k-PP (k-partition problem, $k > 1$): Consider a multiset $S = \{a_1, \cdots, a_m\}$, with

$A = \sum_{i=1}^{m} a_i$ divisible by k. Decide if there exists a partition $S_1, \cdots, S_k$ such that $\sum_{a \in S_j} a = A/k$ for all $1 \leqq j \leqq k$.

k-PP can be transformed into k-TSP as follows: Let $S = \{a_1, \cdots, a_m\}$ be an instance of k-PP. Form a complete graph $G = (V, E)$, where $V = \{v_s, v_1, v_2, \cdots, v_m\}$. 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 $C = (2/k)\sum_{i=1}^{m} a_i$.

k-PP can be reduced to k-CPP as follows: Let $S = \{a_1, \cdots, a_m\}$ 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 $C = (1/k)\sum_{i=1}^{m} a_i$.

k-SCP is shown to be *NP*-complete by reducing k-TSP to it, performing the same transformation as in our reduction from TSP to SCP.

3.1. Approximation algorithms for k-TSP. We shall consider 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, which have been analyzed by Rosenkrantz, et al., [14]. We shall consider generalizations of the nearest neighbor and nearest insertion algorithms to handle k salesmen. We shall find these generalized algorithms to perform rather poorly in worst case.

Let $R = (v_{i_1}, \cdots, v_{i_m})$ 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 $c(u, R) = \min\{c(u, v) | v \in R\}$. The cost of inserting a vertex u between v_{i_p} and $v_{i_{p+1}}$, $1 \leqq p < m$, is $c(v_{i_p}, u) + c(u, v_{i_{p+1}}) - c(v_{i_p}, v_{i_{p+1}})$. 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 $R_1, \cdots, R_k$ be k subtours. We define their cost $c(R_1, \cdots, R_k)$ as the maximum of the costs of each subtour, that is, $c(R_1, \cdots, R_k) = \max_{1 \leqq i \leqq k} \{c(R_i)\}$.

ALGORITHM k-NEARINSERT.

1. Start with k subtours $R_j = (v_s, v_s)$, $1 \leqq j \leqq k$, where v_s is the start vertex.
2. For each j, $1 \leqq j \leqq k$, 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 $c(R_1, \cdots, R_i \leftarrow (u_i), \cdots, R_k)$ is minimum. Insert u_i into the subtour R_i.
4. If $(R_1, \cdots, R_k)$ covers all vertices, then stop. Otherwise go to step 2.

LEMMA 3. *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 $(R_1, \cdots, R_k)$ be a k-tour with $R_i = (v_s, v_{i_1}, \cdots, v_{i_{m_i}}, v_s)$. Let $G_i = (V_i, E_i)$ be the complete subgraph of G, where $V_i = \{v_s, v_{i_1}, \cdots, v_{i_{m_i}}\}$. Suppose the subtour R_i has been built by adding vertices $v_{p(i_1)}, v_{p(i_2)}, \cdots, v_{p(i_{m_i})}$ in this order, where p is a permutation of $(i_1, i_2, \cdots, i_m)$. 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 $c(R_i)/F_i^* < 2$ [14, Thm. 3]. Since $\hat{C}_k = \max_{1 \leqq i \leqq k} \{c(R_i)\}$, and $F_i^* \leqq C_1^*$ for all i, $1 \leqq i \leqq k$, then $\hat{C}_k / C_1^* < 2$. □

THEOREM 2. *If $\hat{C}_k$ 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 3, $\hat{C}_k < 2C_1^*$. By the triangle inequality $C_1^* \leqq kC_k^*$. 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 Fig. 5, which is taken from [14, Thm. 4]. Let the cost of all edges not shown be 2. Let $R_n' = (v_1, v_3, \cdots, v_n, v_{n-1}, v_{n-3}, \cdots, v_1)$ be a tour for the single TSP obtained by 1-NEARINSERT. Then $c(R_n') = 2(n-1)$ and $C_1^* = n$.

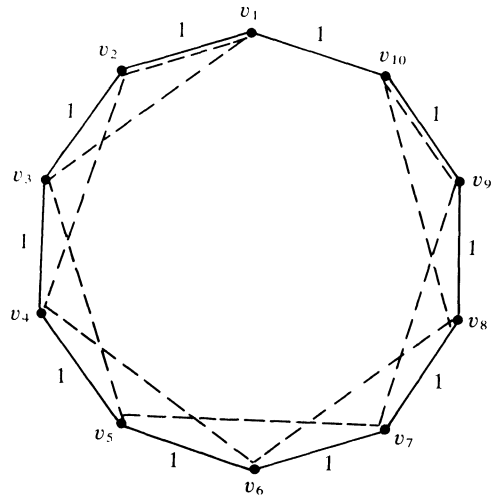

FIG. 5. *G_{10} and a 1-tour by* 1-NEARINSERT.

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 $v_1 = v_{j,1}, v_{j,2}, \cdots, v_{j,kn}$. The cost of edges in $G_{j,kn}$ is the same as in G_{kn}. The cost of edges between vertices in different copies, $v_{j,p}$ and $v_{i,m}$ is 0 if $p = m$; is 1 if $|p-m| = 1$, or if $p = 1$ and $m = kn$; and is 2 otherwise. A k-tour by k-NEARINSERT is $(R_1, \cdots, R_k)$, where for each $1 \leqq j \leqq k$, $R_j = (v_{j,1}, v_{j,3}, \cdots, v_{j,kn}, v_{j,kn-1}, v_{j,kn-3}, \cdots, v_{j,1})$ with $c(R_j) = 2(kn-1)$. Thus $\hat{C}_k = 2(kn-1)$. On the other hand, an optimal k-tour is $(R_1^*, \cdots, R_k^*)$, where $R_1^* = (v_1, v_{1,2}, v_{2,2}, \cdots, v_{k,2}, v_{1,3}, \cdots, v_{k,3}, \cdots, v_{i,n}, \cdots, v_{k,n}, v_1)$, $\cdots$, $R_k^* = (v_1, v_{1,(k-1)n+1}, \cdots, v_{k,(k-1)n+1}, \cdots, v_{1,kn}, \cdots, v_{k,kn}, v_1)$. Thus $c(R_j^*) = n+1$, and $C_k^* = n+1$. Hence

$$C_k/C_k^* = 2(kn-1)/(n+1) = 2k - 2(k+1)/(n+1)$$

which approaches $2k$ for large n. □

The following notation is used in describing the algorithm k-NEARNEIGHBOR. Let $R = (v_{i_1}, \cdots, v_{i_m})$ 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 $R \leftarrow u$.

ALGORITHM k-NEARNEIGHBOR.

1. Set each of the k paths initially to the initial vertex, that is, $R_j = (v_s)$ for all j, $1 \leqq j \leqq k$.
2. For each j, $1 \leqq j \leqq k$, let u_j be the vertex nearest to the terminal vertex of R_j.
3. Let i be such that $c(R_1, \cdots, R_i \leftarrow u_i, \cdots, R_k)$ 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 4. *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. We use the result in [14, Thm. 1];

$$\hat{C}_1 / C_1^* \leqq (1/2) \lceil \log n \rceil + 1/2.$$

A proof similar to that for Lemma 3 then applies. □

THEOREM 3. *If $\hat{C}_k$ is as in Lemma* 4, *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, $C_1^* \leqq kC_k^*$, and by Lemma 4, $\hat{C}_k/(kC_k^*) < (1/2) \log n + 1$. 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 [14, Thm. 2], where $|V| = n = 2^m - 1$. Use an argument similar to that in Theorem 2 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) nonoptimal 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 $\max c(v_1, v_i)$, 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 Fig. 6.

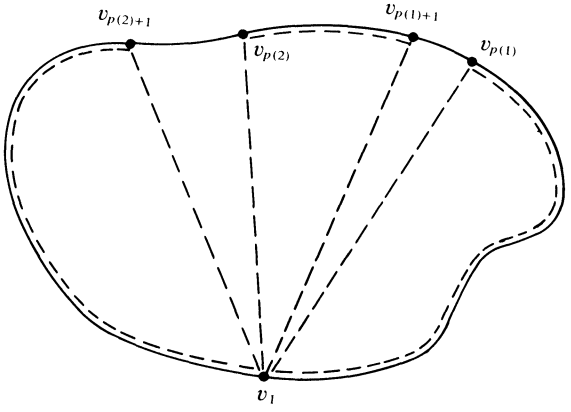

FIG. 6. *Splitting a* 1*-tour into a* 3*-tour.*

ALGORITHM k-SPLITOUR.

1. Find a 1-tour $R=(v_1, v_2, \cdots, v_n, v_1)$ with $c(R)=L$, where v_1 is the initial vertex.
2. For each j, $1 \leqq j < k$, find the last vertex $v_{p(j)}$ such that the cost of the path from v_1 to $v_{p(j)}$ along R is not greater than $(j/k)(L-2c_{\max})+c_{\max}$.
3. Obtain the k-tour by forming k subtours as $R_1=(v_1, \cdots, v_{p(1)}, v_1)$, $R_2=(v_1, v_{p(1)+1}, \cdots, v_{p(2)}, v_1), \cdots, R_k=(v_1, v_{p(k-1)+1}, \cdots, v_n, v_1)$.

THEOREM 4. *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*

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

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

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

$$\begin{aligned}\hat{C}_k = \max c(R_j) &\leqq (1/k)(L-2c_{\max})+2c_{\max} \\ &\leqq (L/k)+2(1-1/k)c_{\max}.\end{aligned}$$

Due to the triangle inequality, $C_k^* \geqq (1/k)C_1^*$ and $c_{\max} \leqq (1/2)C_k^*$. Using these with $L \leqq eC_1^*$, we obtain $\hat{C}_k \leqq (1/k)ekC_k^*+2(1-1/k)(1/2)C_k^*$, which yields $\hat{C}_k/C_k^* \leqq e+1-1/k$. □

COROLLARY 1. *There is an $O(|V|^3)$ approximation algorithm for* k-TSP *with bound*

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

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

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 Fig. 7. All edges shown in Fig. 7(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 Fig. 7(a). Figure 7(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.

3.2. 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 $R=(v_1, e_{i1}, v_{i2}, e_{i2}, \cdots, v_{im}, e_{im}, v_1)$ be the 1-route so obtained. Let $L=c(R)$ and let $R_{v_{in}}$ denote the path $(v_1, e_{i1}, v_{i2}, \cdots, v_{in})$, with $n \leqq m$. We denote the cost of a shortest path from a vertex v to u by $s(v, u)$. Define $s_{\max}$ as $(1/2)\max\{s(v_1, v_{ij})+c(v_{ij}, v_{ij+1})+s(v_{ij+1}, v_1)\}$ and for each j, $1 \leqq j < k$, $L_j=(j/k)(L-2s_{\max})+s_{\max}$. Figure 8 shows the building of a k-route, and Fig. 9 shows a completed 4-route.

ALGORITHM k-POSTMEN.

1. Find an optimal 1-route $R=(v_1, e_{i1}, v_{i2}, \cdots, v_{im}, e_{im}, v_1)$ using the algorithm in [4], where v_1 is the start vertex.
2. For each j, $1 \leqq j \leqq k$, find the last vertex $v_{p'(j)}$ such that $c(R_{v_{p'(j)}}) \leqq L_j$.
3. Let $r_j=L_j-c(R_{v_{p'(j)}})$. For each j, $1 \leqq j < k$, if $r_j+s(v_{p'(j)}, v_1) \leqq c(v_{p'(j)}, v_{p'(j)+1})-r_j+s(v_{p'(j)+1}, v_1)$, then $v_{p(j)}=v_{p'(j)}$. Otherwise $v_{p(j)}=v_{p'(j)+1}$.

4. Let $R_1 = (v_1, e_{i1}, v_{i2}, \cdots, v_{p(1)})$, $R_2 = (v_{p(1)}, \cdots, v_{p(2)}), \cdots, R_k = (v_{p(k-1)}, \cdots, v_1)$. 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.

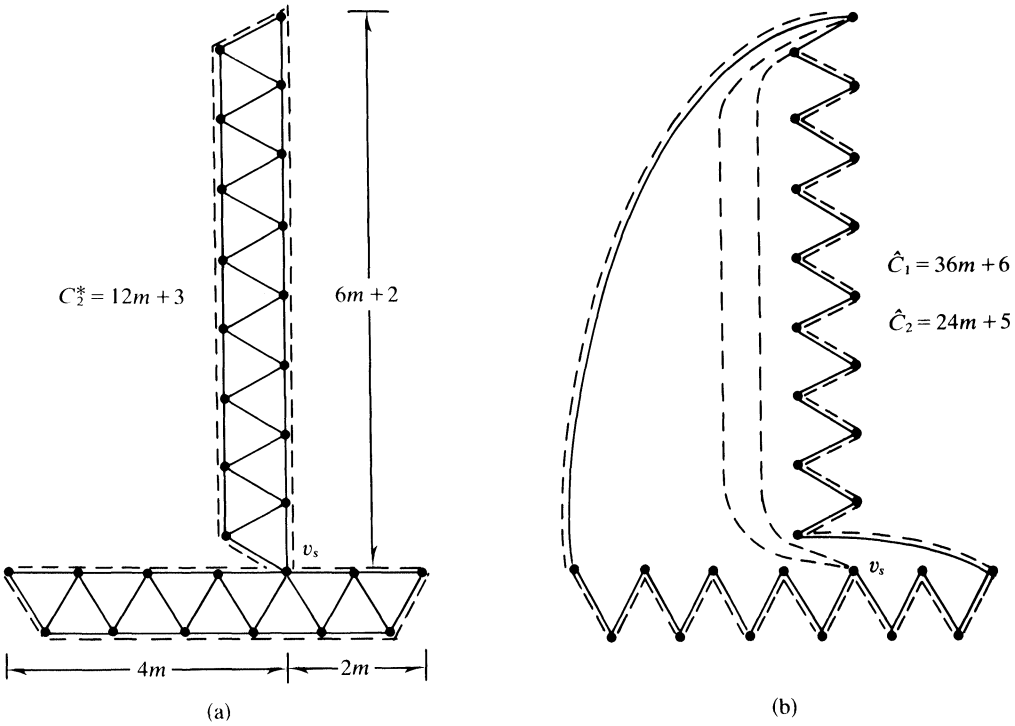

FIG. 7. *Worst case for* 2-TSP.

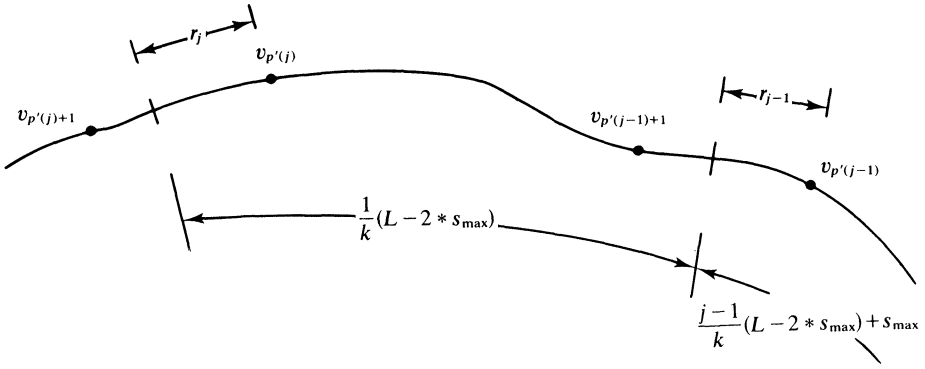

FIG. 8. *Building a k-route.*

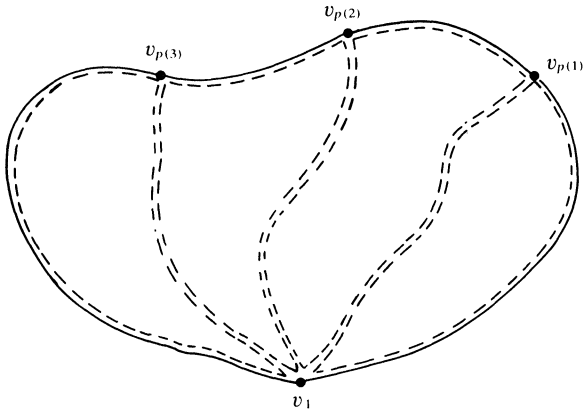

FIG. 9. *A 4-route.*

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 in which subtour to include the edge.

THEOREM 5. *The algorithm* k-POSTMEN *produces* $\hat{C}_k$ *in* $O(|V|^3)$ *time such that*

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

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

$$s(v_1, v_{p'(j)})+c(v_{p'(j)}, v_{p'(j)+1})+s(v_{p'(j)+1}, v_1) \leqq 2s_{\max}.$$

Hence

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

Similarly,

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

The worst case for the jth subroute R_j is when it starts from v_1, reaches $v_{p'(j-1)}$, continues to $v_{p'(j)+1}$ along R and finally back to v_1. But in this case $s(v_1, v_{p'(j-1)})+r_{j-1} \leqq s_{\max}$ and $c(v_{p'(j)}, v_{p'(j)+1})-r_j+s(v_{p'(j)+1}, v_1) \leqq s_{\max}$. Thus $c(R_j) \leqq s_{\max}+(1/k)(L-2s_{\max})+s_{\max}$. Since $L \leqq kC_k^*$ and $s_{\max} \leqq (1/2)C_k^*$,

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

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

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. $R=(v_1, v_{i2}, \cdots, v_{im}, v_1)$, $R_{v_{in}}$, L, $c_{\max}$ and $L_j=(j/k)(L-2c_{\max})+c_{\max}$ are as previously defined.

ALGORITHM k-CRANES.

1. Find a 1-tour $R=(v_1, v_{i2}, \cdots, v_{im}, v_1)$ for the single crane problem with $c(R)=L$, where v_1 is the initial vertex.
2. For each j, $1 \leqq j < k$, find the last vertex $v_{p'(j)}$ such that $c(R_{v_{p'(j)}}) \leqq L_j$.

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

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

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

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

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

4. For each j, $1 \leqq j \leqq k$, 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 6. *Algorithm* k-CRANES *produces* $\hat{C}_k$ *such that*

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

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

Proof. A proof which is a combination of the proofs of Theorems 4 and 5 applies. □

COROLLARY 2. *There is an approximation algorithm of* $O(\max\{|V|^3, |A|^3\})$ *time complexity for* k-SCP *such that*

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

Proof. In Step 1, a 1-tour may be obtained by using the algorithm in § 2, for which $e = 9/5$. □

4. Conclusion. In developing an approximation algorithm for the stacker-crane problem, we have applied a mixed strategy approach. Our algorithm consists of two efficient algorithms, each of which handles certain extreme cases well, so that the overall behavior of the algorithm is improved. We have formulated k-person routing problems as minimax problems. We have presented a tour-splitting heuristic, a method by which approximate solutions for multirouting problems can be obtained from good approximate solutions of simple routing problems.

REFERENCES

[1] M. BELLMORE AND S. HONG, *Transformation of the multisalesmen problem to the standard traveling salesman problem*, J. Assoc. Comput. Mach., 21 (1974), pp. 500–504.

[2] N. CHRISTOFIDES, *Worst-case analysis of a new heuristic for the traveling salesman problem*, Management Sciencies Research Report No. 388, Carnegie–Mellon University, Pittsburgh, PA, (1976).

[3] S. A. COOK, *The complexity of theorem proving procedures*, 3rd Symposium on Theory of Computing (1971), pp. 151–158.

[4] J. EDMONDS AND E. L. JOHNSON, *Matching, Euler tours and the Chinese postman*, Math. Programming, 5 (1973), pp. 88–124.

[5] R. FLOYD, *Algorithm 97, shortest path*, Comm. ACM, 5 (1962), p. 345.

[6] H. GABOW AND E. L. LAWLER, *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.

[7] M. R. GAREY AND D. S. JOHNSON, *Approximation Algorithms for Combinatorial Problems: An Annotated Bibliography*, Algorithms and Complexity: Recent Results and New Directions, J. F. Traub, ed., Academic Press, New York, 1976, pp. 41–52.

[8] O. H. IBARRA AND C. E. KIM, *Fast approximation algorithms for the knapsack and sum of subset problems*, J. Assoc. Comput. Mach., 22 (1975), pp. 463–468.
[9] D. S. JOHNSON, *Approximation algorithms for combinatorial problems*, J. Comput. Systems Sci., 9 (1974) pp. 256–278.
[10] R. M. KARP, *Reducibility among combinatorial problems*, Complexity of Computer Computations, R. E. Miller and J. W. Thatcher eds., Plenum Press, New York, 1972, pp. 85–104.
[11] C. E. MILLER, A. W. TUCKER AND R. A. ZEMLIN, *Integer programming formulation of traveling salesman problem*, J. Assoc. Comput. Mach., 7 (1960), pp. 362–329.
[12] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Some complexity results for the traveling salesman problem*, 8th Symposium on Theory of Computing, 1976, pp. 1–9.
[13] D. J. ROSENKRANTZ, private communication, 1976.
[14] D. J. ROSENKRANTZ, R. E. STEARNS AND P. M. LEWIS, *Approximate algorithms for the traveling salesperson problem*, 15th Symposium on Switching and Automata Theory (1974), pp. 33–42.
[15] S. K. SAHNI, *Approximate algorithms for the* 0/1 *knapsack problem*, J. Assoc. Comput. Mach., 22 (1975), pp. 115–124.
[16] S. K. SAHNI AND T. GONZALEZ, *P-complete approximation problems*, J. Assoc. Comput. Mach., 23 (1976), pp. 555–565.