Approximation Algorithms for Some Postman Problems

GREG N. FREDERICKSON

University of Maryland, College Park, Maryland

ABSTRACT Approximation algorithms for several NP-complete edge-covering routing problems are presented and analyzed in terms of the worst-case ratio of the cost of the obtained solution to the cost of the optimum solution A worst-case bound of 2 is proved for the mixed postman algorithm of Edmonds and Johnson, and a related algorithm for the mixed postman problem is shown also to have a worst-case bound of 2 A mixed strategy approach is used to obtain a bound of $\frac{5}{3}$ for the mixed postman problem A second mixed strategy algorithm, for the mixed postman on a planar graph, is shown to have a worst-case bound of $\frac{3}{2}$

KEY WORDS AND PHRASES Chinese postman problem, mixed postman problem, rural postman problem, NP-complete problems, polynomial-time approximation algorithm, heuristic, worst-case performance bound, mixed strategy

CR CATEGORIES 5 25, 5 39, 8 3

1. *Introduction*

Routing problems can be used to model practical tasks such as mail delivery, trash collection, parcel pickup and delivery, and snow removal. While some routing problems are efficiently solvable, many are NP-complete in the sense of Cook [3] and Karp [13]. The edge-covering problems that we consider in this paper, the mixed postman problem and the rural postman problem, are both NP-complete. No efficient algorithm yielding exact solutions has been discovered for any of the NP-complete problems, so that a reasonable alternative is to find algorithms that are efficient but yield only approximate solutions [10–12, 21]. We shall consider algorithms for which worst-case analysis has been performed

Rosenkrantz, Stearns, and Lewis [20] have applied worst-case analysis to several algorithms for the traveling salesman problem, in which a tour is built incrementally. Christofides [2] has developed an algorithm for the traveling salesman problem with a better worst-case bound by using a global approach with respect to the graph. Frederickson, Hecht, and Kim [8] applied a mixed strategy approach to the stacker-crane problem, a variant of the traveling salesman problem. Their approach uses two algorithms, each of which handles certain extreme cases well, and chooses the better of the two results. We shall use a global approach in all of our approximation algorithms and a mixed strategy approach in two of them.

The Chinese postman problem requires that all edges in an undirected graph be included at least once in a tour of minimum cost. The problem was proposed by Mei-Ko [16] and shown to be efficiently solvable by Edmonds [4]. If all edges in the graph are directed, then the problem is also efficiently solvable, as shown by Edmonds and Johnson [5]. But if the graph is mixed (that is, contains both directed and undirected edges), then the problem has been shown by Papadimitriou [19] to be NP-complete. Practical special cases of the

This research was partially supported by the National Science Foundation under Grant DCR-08361

Author's present address Computer Science Department, The Pennsylvania State University, University Park, PA 16802

problem, including the case in which the graph is planar, have also been shown by Papadimitriou to be NP-complete.

Edmonds and Johnson [5] have presented an approximation algorithm for the mixed postman problem but have not bounded its worst-case performance. In Section 3, we prove that their algorithm has an approachable worst-case bound of 2 In Section 4, we present a related algorithm for the mixed postman problem which also has a worst-case bound of 2. In Section 5, we combine these two algorithms into a mixed strategy algorithm with a worst-case bound of $\frac{5}{3}$. For the special case of a planar graph, in Section 6 we develop an algorithm with a worst-case bound of $\frac{3}{2}$.

The rural postman problem, as described in Orloff [17], is a variant of the Chinese postman problem in which only a subset of the edges in the graph must be included in a tour Lenstra and Rinnooy Kan [15] have shown that the rural postman problem is NP-complete. Orloff [18] observes that while the rural postman problem is NP-complete, the determining factor in the complexity of the problem seems to be the number of disconnected components in the required edge set. We note the existence of an exact recursive algorithm that is exponential only in the number of disconnected components. We also note that a $\frac{3}{2}$ approximation algorithm is obtainable by using an algorithm similar to the traveling salesman algorithm of Christofides [2]. Neither result is especially difficult, and we omit further reference to the rural postman problem.

2. *Definitions*

In this paper we shall consider graphs that may have more than one edge between two vertices. An *undirected graph* $G = (V, E)$ consists of a set V of vertices and a multiset 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 multiset E of undirected edges, and a multiset 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 $\langle t, h\rangle$. An *oriented edge* $\langle u, v\rangle$ is an undirected edge (u, v) that has been given direction from u to v. A cost function will be defined on $E \cup A$, with all costs nonnegative.

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. We shall use the normal set operators in operations involving multisets, with the result being 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 of *odd degree* otherwise. The *parity* of a vertex is whether it is of even or odd degree.

Given a mixed graph $G = (V, E, A)$, a *path* is a sequence of edges and arcs, such that if $[u, v]$ and $[w, x]$ are two adjacent edges or arcs in the tour, then $v = w$. A *cycle* is a path such that if $[u, v]$ is the first edge or arc and $[w, x]$ is the last edge or arc, then $u = x$ A *tour* is a cycle that includes all edges and arcs in G. An arc $\langle t, h\rangle$ is *traversed* by covering it in a direction from its tail t to its head h. The *cost of a tour* is the total of the costs for each instance of the arcs and edges in the tour.

A mixed graph is *planar* if it can be realized in the plane, that is, if the edges and arcs can be mapped into a set of nonintersecting curves in the plane. We shall consider a planar graph and its realization in the plane to be interchangeable for ease in definitions. A *face* of a planar graph is a continuous area that is enclosed by edges and arcs and that contains no edge or arc. Two cycles are *noncrossing* if the area inside one cycle is either completely inside or outside the other cycle The *clockwise adjacent* edge of an edge (u, v) incident on a vertex v is the edge (v, w), which is the next edge around v in a clockwise direction. The *handedness* of an edge with respect to a cycle refers to whether the edge is inside or outside the cycle.

A (maximum cardinality) *matching* $E' \subseteq E$ of a graph $G = (V, E)$ is a (maximum) subset of edges that 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 $G' = (V, (V_1 \times V_2) \subseteq E)$. A *spanning tree* of a connected graph $G = (V, E)$ is a tree $T = (V, E')$ where $E' \subseteq E$. A *minimum-cost spanning tree* is a spanning tree such that the total cost of the edges E' is minimum.

3. *The Edmonds and Johnson Mixed Postman Algorithm*

For the Chinese postman problem on an undirected graph, there exists a tour using each edge of the graph just once if the following property is satisfied: The degree of each vertex must be even. For the Chinese postman problem on a directed graph, the following property must be satisfied: For each vertex in the graph, the indegree of arcs adjacent to it must equal the outdegree. For a mixed graph, if both properties are satisfied, then a tour exists. (See Ford and Fulkerson [7] for a precise statement of necessary and sufficient conditions.) Thus a method of solution for the Chinese postman problem consists of finding a minimum-cost augmentation of the graph that satisfies the sufficient properties, and then identifying the tour over the augmented graph.

The Edmonds and Johnson algorithm for the mixed postman problem essentially consists of two steps. The first step is to modify the graph to make the degree of each vertex even, treating each arc as an edge. The second step is to make the indegree of each vertex equal to the outdegree, while preserving the property that each vertex has even degree. A tour can be easily found from the resulting graph. We specify the Edmonds and Johnson algorithm and its subroutines below.

Algorithm MIXED1

Input A mixed graph $G = (V, E, A)$ and cost function c on $E \cup A$

Output A tour that covers all edges and arcs in G

1 Call EVENDEGREE
2 Call INOUTDEGREE with the output from EVENDEGREE
3 Call EVENPARITY with the output from INOUTDEGREE
4 Find an orientation for the undirected edges in the resulting graph, and then a traversal of all arcs and oriented edges

Algorithm EVENDEGREE

Input A mixed graph $G = (V, E, A)$ with cost function c on $E \cup A$

Output A mixed graph $G' = (V, E', A')$ where $E \subseteq E'$, $A \subseteq A'$, and the degree of each vertex, ignoring arc direction, is even

1 Identify the vertices of odd degree
2 Find all shortest paths between the odd vertices, ignoring arc direction
3 Perform a minimum-cost matching of the odd vertices using the shortest path distances
4. Set multiset A' to A and multiset E' to E Insert arcs in the shortest paths used in the matching into A' and edges in the shortest paths used in the matching into E'

Algorithm INOUTDEGREE makes additional copies of arcs and edges and orients some edges, so as to make the indegree and outdegree of each vertex equal. Edmonds and Johnson have formulated the problem in terms of a network minimum-cost flow problem, in which the cost of the additional arcs and edges is minimized. Sets E_1 and E_2 of oriented edges corresponding to set E are introduced. Set E_2 will be used to determine the orientation of those edges that are oriented by Algorithm INOUTDEGREE. Set E_1 will be used to determine how many additional copies of an oriented edge are required.

Algorithm INOUTDEGREE

Input Mixed graph $G = (V, E, A)$ with cost function c

Output A multiset M that contains every arc of G at least once and oriented edges, such that for each vertex the indegree equals the outdegree Also, a set $U \subseteq E$ of edges that have not been oriented and inserted into M

1 Let E_1 be a multiset of oriented edges, such that for each edge (v, w) in E there are two oriented edges $\langle v, w\rangle$ and $\langle w, v\rangle$ in E_1 Let E_2 be composed in the same manner Let $E_t = E_1 \cup E_2 \cup A$

2 For each vertex v, let b_v be the number of arcs in A directed toward v minus the number of arcs in A directed away from v

3 Minimize z subject to

$$z = \sum_{s \in A} c_s x_s + \sum_{s \in E_1} c_s x_s,$$

x_s a nonnegative integer for $s \in A \cup E_1$,

$x_s = 0$ or 1 for $s \in E_2$,

$$\sum \{x_s \mid s \in E_t \text{ is directed away from } v\} - \sum \{x_s \mid s \in E_t \text{ is directed toward } v\} = b_v,$$

where x_s is the number of additional copies of arc s from A, or the number of copies of oriented edge s from E_1 or E_2

4 Initialize U to be empty and M equal to A For each arc s in A, insert x_s additional copies of s into M For each oriented edge s in E_1, insert x_s copies of s into M

5 For each pair of corresponding oriented edges $s_1 = \langle v, w\rangle$ and $s_2 = \langle w, v\rangle$ in E_2 If $x_{s_1} + x_{s_1} = 1$, then insert x_{s_1} copies of s_1 and x_{s_2} copies of s_2 into M Otherwise, insert (v, w) into U

LEMMA 1. *The time complexity of Algorithm INOUTDEGREE is no worse than* $O(|A|(max\{|A|, |E|\})^2)$.

PROOF. The minimum-cost network flow problem can be solved by the out-of-kilter method, as described in [14]. The time complexity will be bounded by $O(mK)$, where m is the number of directed edges in the network, and K is the sum of the kilter numbers for the initial circulation. If the initial circulation is zero, then the kilter number of an edge will be no larger than the capacity of the edge, which need be no larger than the total demand $|A|$. Thus the sum K of kilter numbers will be at worst $O(|A|m)$. The number of directed edges in the network will be $m = O(\max\{|A|, |E|\})$. Thus the time complexity will be $O(|A|m^2) = O(|A|(\max\{|A|, |E|\})^2)$. □

Edmonds and Johnson show that there is a minimum-cost solution to their network flow constraints that preserves the property that each vertex has even degree. However, we note that the network flow constraints are not sufficient to select one of the solutions that preserves even degree. Their argument merely demonstrates the existence of a minimum-cost solution that preserves even degree. Thus Algorithm INOUTDEGREE does not necessarily maintain even degree. As an example, consider the graph in Figure 1 (a). An augmentation that might be produced by Algorithm INOUTDEGREE and that would satisfy the Edmonds and Johnson constraints is shown in Figure 1 (b). It clearly does not preserve even degree. We now specify Algorithm EVENPARITY, for which the cost of additional arcs and edges will not increase. The output of EVENPARITY will be a graph

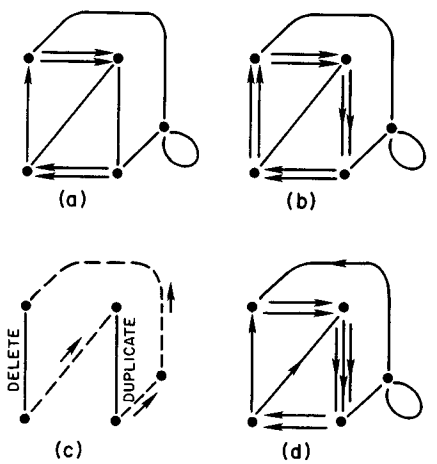

FIG 1 Algorithm EVENPARITY applied to a graph

satisfying the properties that at each vertex the degree is even, and the indegree equals the outdegree.

Algorithm EVENPARITY

Input Graph $G = (V, E, A)$ and sets M and U, which are output from Algorithm INOUTDEGREE

Output Sets M' and U' satisfying the same criteria as sets M and U from INOUTDEGREE, such that vertices in (V, U', M') are of even degree

1 Identify the set of odd vertices V' with respect to (V, U, M) Set M' to M, and U' to U
2 Let $M'' \subseteq M$ be the set of additional arcs and oriented edges created by INOUTDEGREE
3 Call ADJUSTCYCLES

Algorithm ADJUSTCYCLES identifies cycles consisting of alternating paths in M'' and U, with each path anchored at each end by an odd vertex. The paths in M'' will be formed without considering the direction of the arcs and oriented edges. As the cycles are covered, arcs or oriented edges on the cycles will be either duplicated or deleted, and edges on the cycles will be oriented. This is done such that the parity of odd vertices is changed, while maintaining indegree equal to outdegree for each vertex.

Algorithm ADJUSTCYCLES

```
1  While V' is not empty
2      Select v from V', and set v_s to v
3      While v is in V'·
4          Remove v from V'
5          Repeat
               Remove [v, w] from M''
               If [v, w] is directed toward w,
               Then insert a duplicate copy of ⟨v, w⟩ into M',
               Otherwise delete a copy of ⟨w, v⟩ from M'
               Set v to w
6          Until v is in V'
7          Remove v from V'
8          Repeat
               Remove (v, w) from U' Orient (v, w) from v to w and insert it into M' Set v to w
9          Until v is in V' or v = v_s
```

LEMMA 2. *Let M and U be the output of Algorithm INOUTDEGREE for which the input graph is G', a graph with every vertex having even degree. Then Algorithm EVENPARITY will output M' and U' of total cost equal to that of M and U, such that each vertex has even degree with respect to $M' \cup U'$.*

PROOF. Let V' and M'' be the same as in Algorithm EVENPARITY. We see that there are sufficient edges, arcs, and oriented edges for ADJUSTCYCLES to identify cycles. Each odd vertex will have an odd number of elements from M'' incident on it. This is because the only way that Algorithm INOUTDEGREE can change a vertex from odd to even is the addition of extra copies of arcs and oriented edges. Further, every vertex in V' will have an odd number of edges from U incident on it. Otherwise the vertex would be even, since indegree equals outdegree from Algorithm INOUTDEGREE. By similar reasoning, an even vertex will have an even number of elements from M'' incident on it, and an even number of edges from U incident on it. These properties allow ADJUSTCYCLES to complete paths in U from one odd vertex to another. The same applies for paths in M''.

Algorithm EVENPARITY will change the parity of odd vertices only. Step 5 is applied to an even number of the arcs and oriented edges incident with an even vertex, and to an odd number of the arcs and edges incident with an odd vertex.

Algorithm EVENPARITY will not change the property that indegree equals outdegree

at each vertex. Adjustment of M' in one iteration of either step 5 or 8 of ADJUSTCYCLES will cause w to temporarily have a difference of indegree minus outdegree of +1. However, this is always corrected on the next iteration of either step 5 or 8.

M' and U' will cost no more than M and U. Suppose the cost of additional copies of edges and arcs in M' is greater than that in M. Then reverse the direction of each edge taken from U, insert two copies of each arc or oriented edge that was deleted, and delete two copies of each edge or arc that was inserted. This new augmentation must be of less cost than that of M and U. This is a contradiction, since M and U were created by a minimum-cost network flow algorithm. Thus M' and U' cost no more than M and U. M' and U' will not cost less than M and U, since M and U are produced by INOUTDEGREE, which minimizes the cost of additional arcs and edges. Thus the cost of M' and U' equals the cost of M and U. □

We consider an example of EVENPARITY applied to the graph in Figure 1 (b), which might be the result of applying Algorithm INOUTDEGREE to the graph in Figure 1 (a). Assume each edge and arc has unit cost. Figure 1 (c) shows the odd vertices, with the paths of edges connecting them shown in dashed lines and the paths of arcs and oriented edges shown in solid lines. In this example there is just one cycle. If the vertex marked v_s is the first vertex examined, then the arcs and edges will be adjusted as indicated in Figure 1 (c). Figure 1 (d) shows the final modified graph.

We now bound the worst-case behavior of the Edmonds and Johnson algorithm. While Algorithms EVENDEGREE and INOUTDEGREE each modify their input graph in minimum-cost fashion, we find that the algorithm as a whole does not necessarily produce an optimum solution.

THEOREM 1. *If C^* is the cost of an optimum tour for the mixed postman problem, and $\hat{C}$ is the cost of a tour generated by Algorithm MIXED1, then*

$$\hat{C}/C^* \leq 2$$

and the bound is approachable. The time complexity of Algorithm MIXED1 is no worse than $O(max\{|V|^3, |A|(max\{|A|, |E|\})^2\})$.

PROOF. Let G_2 be the graph that results from duplicating every arc and edge of G. Let C_2 be the cost of a solution by the algorithm on input G_2. Since the degree of each vertex in G_2 is even, only the indegree and outdegree of each vertex must be made equal, and INOUTDEGREE does this optimally. Hence the cost C_2 produced by the algorithm on G_2 is optimal. Now since doubling an optimum solution for G is a feasible solution for G_2, we have $C_2 \leq 2C^*$.

Let G_1 be the graph that results from applying EVENDEGREE to G. Since EVENDEGREE employs a minimum-cost matching, no edge will be duplicated more than once (else two duplicate copies could be discarded for a matching cheaper than the minimum-cost one). Thus the sets of edges and arcs of G_1 are subsets of the sets of edges and arcs of G_2. Since the degree of every vertex in G_1 and G_2 is even, the algorithm applied to G_1 and G_2 will yield costs $\hat{C}$ and C_2, respectively, which are optimal with respect to G_1 and G_2 The cost of an optimum solution to G_1 cannot be greater than the cost of an optimum solution to G_2. Hence $\hat{C} \leq C_2$. Combining this result with that of the preceding paragraph, we have $\hat{C} \leq C_2 \leq 2C^*$.

To see that the bound is approachable, consider the graph in Figure 2 (a). Algorithm EVENDEGREE will duplicate the two arcs. However this will force the two edges of cost 1 to be duplicated by Algorithm INOUTDEGREE, as shown in Figure 2 (b). An optimum solution in Figure 2 (c) will cover each of the edges only once. As ϵ goes to zero, the bound will approach 2.

Algorithm EVENDEGREE will require at most $O(|V|^3)$ time, since it is dominated by the all shortest paths algorithm [6] and the minimum-cost matching algorithm [9]. The graph G' output by EVENDEGREE will have $|A'| = O(|A|)$ and $|E'| = O(|E|)$. Thus the complexity of INOUTDEGREE will be the same as in Lemma 1. □

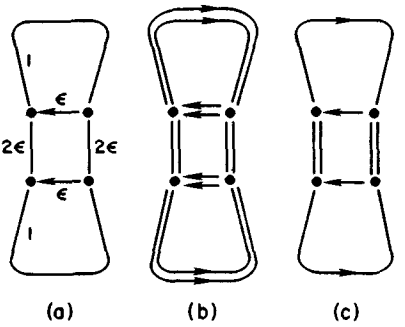

FIG 2 A worst-case example for Algorithm MIXED1

4. *A Second Algorithm for the Mixed Postman Problem*

An alternative approach to the mixed postman problem is the reverse of the Edmonds and Johnson approach. First the indegree and outdegree of each vertex are made equal. Then the degree of each vertex is made even, while preserving the property that for each vertex the indegree equals the outdegree. We now give an algorithm that makes use of Algorithm INOUTDEGREE to also achieve a worst-case bound of 2 for the mixed postman problem.

Algorithm LARGECYCLES

Input Output of INOUTDEGREE

Output A tour covering all edges and arcs of G

1 Identify vertices of odd degree in the graph $G' = (V, U)$
2 Find all shortest paths between the odd vertices over the graph $G'' = (V, E)$
3 Perform a minimum-cost matching of the odd vertices using the shortest path distances Insert the edges used in the matching into U
4 Find a traversal of the arcs and oriented edges in M and unoriented edges in U

Algorithm MIXED2

Input A mixed graph $G = (V, E, A)$ and cost function c on $E \cup A$

Output A tour that covers all edges and arcs in G

1 Call INOUTDEGREE.
2 Call LARGECYCLES

THEOREM 2. *If C^* is the cost of an optimum tour for the mixed postman problem, and $\hat{C}$ is the cost of a tour generated by Algorithm MIXED2, then*

$$\hat{C}/C^* \leq 2,$$

and the bound is approachable. The time complexity is the same as for Algorithm MIXED1.

PROOF. The cost of the edges and arcs in set M found by Algorithm INOUTDEGREE plus the cost of edges in U is at most C^*. The matching in step 3 of LARGECYCLES will produce a set of edges in which no edge appears more than once. Thus the cost of the matching edges will be at most C^*, and the total cost of Algorithm MIXED2 will be at most $2C^*$.

To see that the bound is approachable, consider the graph in Figure 3 (a). Algorithm INOUTDEGREE may match the arcs into a cycle of cost 2ϵ, thus adding no arcs or oriented edges to the graph in Figure 3 (a). Step 3 of LARGECYCLES will duplicate each edge at a total edge cost of 4, as shown in Figure 3 (b). An optimum tour will cover each edge just once, while covering one of the arcs twice, at a total cost of $2 + 3\epsilon$, as shown in Figure 3 (c). □

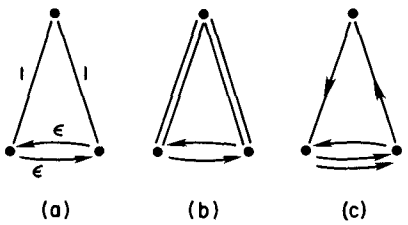

FIG 3 A worst-case example for Algorithm MIXED2

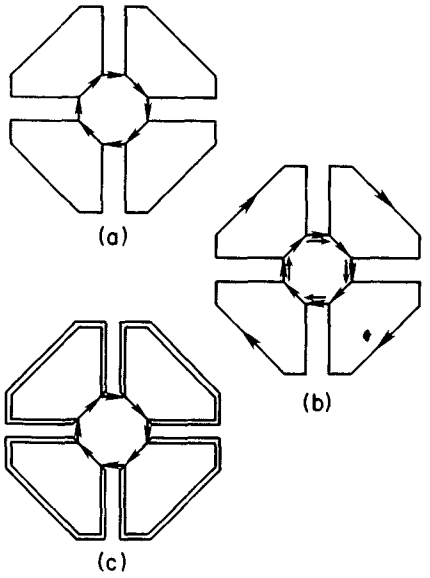

FIG 4 A worst-case example for a revised distance function

If we examine the worst-case example for Algorithm MIXED2 in Figure 3, we might conclude that the algorithm performs poorly because of the distance function used in the matching of LARGECYCLES. The cost of connecting the two vertices with a path directed in either direction is just ϵ, and the distance function should take this into account. We define a more "informed" distance function and show that in the worst case it performs no better. Define the distance as

$$c'(v, w) = \min\{c(v, w), \max\{c\langle v, w\rangle, c\langle w, v\rangle\}\}$$

where $c\langle v, w\rangle$ is the cost of the shortest directed path from v to w in $E \cup A$, and $c(v, w)$ is, as before, the cost of the shortest path from v to w in E.

Such a distance function will allow Algorithm MIXED2 to generate an optimum solution for the example in Figure 3. The matching "edge" is inserted into U. When a traversal is determined, if the "edge" consists of arcs, then the arcs consistent with the traversal direction are used instead. While the distance function performs quite well on this example, it can do poorly on other examples. Consider the graph in Figure 4 (a). The optimum solution duplicates the small arcs and is shown in Figure 4 (b). The revised distance function will not use the cost of the small arcs, but rather the cost of the "petal" edges. The obtained solution is shown in Figure 4 (c). With suitable choice of costs, the example approaches a worst-case bound of 2.

A similar problem may be seen with Algorithm MIXED1. Allowing arc distances to be used in the matching of step 1 of MIXED1 may allow problems of the sort shown in Figure 2. If the matching in step 1 is done using the same revised distance function, then the problem in Figure 2 is overcome. However, the algorithm would then be similar to the modified MIXED2. The worst-case example in Figure 4 will also yield poor behavior for Algorithm MIXED1.

5. *A Mixed Strategy Algorithm for the Mixed Postman Problem*

If we examine the worst-case examples for Algorithms MIXED1 and MIXED2, we discover the following: Algorithm MIXED2 will produce an optimum tour for the worst-case example for MIXED1 in Figure 2. Algorithm MIXED1 will produce an optimum tour for the worst-case example for MIXED2 in Figure 3. This behavior suggests that each

algorithm handles well extreme cases that the other algorithm does not handle well Therefore we shall consider a mixed strategy approach employing both algorithms.

The following notation shall be used in the analysis of the worst-case behavior of the algorithm. Let C^* represent the cost of an optimum tour, and let C_M be the total cost of edges and arcs in set M produced by Algorithm INOUTDEGREE on the original graph. We now analyze MIXED1 and MIXED2 more carefully, in terms of both C^* and C_M.

LEMMA 3. *The cost $\hat{C}$ of a tour generated by Algorithm MIXED2 is at most $2C^* - C_M$*

PROOF. The cost of the edges and arcs in set M from Algorithm INOUTDEGREE is C_M, and the cost of edges not in M is at most $C^* - C_M$. Since in the worst case each edge in U will be duplicated, the cost of the undirected edges after the matching is at most $2(C^* - C_M)$. Thus the total cost is at most $2C^* - C_M$. □

LEMMA 4. *If C_M is the cost of arcs and oriented edges in the set M that is output from Algorithm INOUTDEGREE performed on the original graph G, then the cost of Algorithm MIXED1 is no more than $C^* + 2C_M$*

PROOF. Let G_1 be the graph that results from applying EVENDEGREE to G. Let E' be the set of undirected edges in G_1. The total cost C_E of E' is no larger than the cost of all arcs and edges in G_1, which is no larger than C^*.

From the proof of Theorem 1 we know that G_1 contains at most one duplicate of every original arc of G. Let M_1 be the multiset of arcs and oriented edges such that for every element in M there are exactly two copies of it in M_1. M achieves equal in- and outdegree at every vertex in G; hence M_1 achieves equal in- and outdegree at every vertex in G_1. Therefore, the cost of all arcs and edges in the graph resulting from applying INOUTDEGREE to G_1 is bounded by $2C_M + C_E$. Steps 3 and 4 of MIXED1 do not change the total cost; hence the total cost of MIXED1 is bounded by $C^* + 2C_M$. □

If the cost C_M is large relative to C^*, then Lemma 3 indicates that Algorithm MIXED2 is an appropriate approach. If the cost of set M, C_M, is small relative to C^*, then Lemma 4 indicates that MIXED1 is an appropriate approach. By performing both MIXED1 and MIXED2 and then choosing the better result, we can guarantee a worst-case bound less than 2.

Algorithm GENERALMIXED

Input A weighted, mixed graph G

Output A tour that covers all edges and arcs in G

1 Call MIXED1.
2 Call MIXED2
3 Select the tour of smaller cost.

THEOREM 3. *Algorithm GENERALMIXED produces a tour such that*

$$\hat{C}/C^* \leq \tfrac{5}{3}$$

The time complexity of GENERALMIXED is the same as that for Algorithm MIXED1.

PROOF. If $C_M > \frac{1}{3}C^*$, then consider the result from Lemma 3 concerning Algorithm MIXED2:

$$\begin{aligned}\hat{C}/C^* &\leq (2C^* - C_M)/C^* \\ &< (2C^* - \tfrac{1}{3}C^*)/C^* \\ &= \tfrac{5}{3}.\end{aligned}$$

If $C_M \leq \frac{1}{3}C^*$, then consider the result from Lemma 4 concerning Algorithm MIXED1:

$$\begin{aligned}\hat{C}/C^* &\leq (2C_M + C^*)/C^* \\ &\leq (\tfrac{2}{3}C^* + C^*)/C^* \\ &= \tfrac{5}{3}.\end{aligned}$$ □

We have found no example for Algorithm GENERALMIXED with a worst-case bound

of $\frac{5}{3}$. The closest that we have been able to come is $\frac{3}{2}$, which can be achieved with a graph that is a combination of Figures 2 and 3. However, we have not been able to tighten the result from Lemma 4 so as to give us a bound of $\frac{3}{2}$.

6. *An Approximation Algorithm for the Planar Mixed Postman*

We now consider a second mixed strategy approach, which will work in the case that the graph is planar. The approach is based on performing Algorithm INOUTDEGREE first, and then two algorithms that handle extreme cases. We recall that the arcs and edges in set M can be viewed as forming cycles, since within set M each vertex will have indegree equal to outdegree. If the cost of the cycles C_M is large relative to C^*, then Algorithm LARGECYCLES is used. Algorithm LARGECYCLES will do no worse than duplicate the edges not in cycles. If C_M is small relative to C^*, then the problem can be mapped to a Chinese postman problem on an undirected graph by shrinking the cycles to nodes. Certain portions of the cycles must be duplicated when the nodes are reconstituted into cycles, so that the indegree and outdegree of vertices will remain equal. We now develop Algorithm SMALLCYCLES to handle this second case

Algorithm SMALLCYCLES

Input Output of INOUTDEGREE

Output A tour covering all edges and arcs of G

1 Call GETCYCLES (which partitions M into sets of noncrossing cycles)
2 Call GETREGIONS (which partitions U into sets of edges and specifies the cycles that form boundaries)
3 Initialize M' to be empty For each region, call HANDLEREGION, and insert the resulting multiset of edges and arcs into M'
4 Find a traversal of the arcs and oriented edges in the multiset $M' \cup M$

Algorithm INOUTDEGREE returns a multiset of edges and arcs from which cycles can be extracted. However, it is important in the analysis of the algorithm for the cycles to be noncrossing. This will ensure that no cycle forms the boundary of more than two regions.

Algorithm GETCYCLES

Input A nonempty multiset M of edges and arcs of planar graph G, such that for each vertex the indegree equals the outdegree

Output A set of noncrossing cycles consisting of edges and arcs from M

1. For any arc or edge that has been duplicated, explicitly introduce multiple copies of the edge or arc into the graph, so as to preserve planarity
2 Set p, a counter for the number of cycles, to 0
3 While M is not empty
4 Increment p Extract $\langle u, v\rangle$, an arc or oriented edge from M Insert $\langle u, v\rangle$ into cycle(p)
5 While the degree of v relative to M is odd
6 Find $\langle v, w\rangle$, the arc or oriented edge directed out of v, such that for edges and arcs incident on v between $\langle u, v\rangle$ and $\langle v, w\rangle$, the indegree equals the outdegree, and $\langle v, w\rangle$ is the first such arc in a clockwise direction Extract $\langle v, w\rangle$ from M and insert it into cycle(p) Set u to v and set v to w

An example of how arcs at a vertex are matched in step 6 of GETCYCLES is shown in Figure 5. Each arc and its mate is labeled with the same number. The cycles can be viewed as noncrossing, as shown on the right.

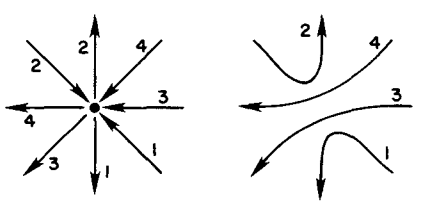

FIG 5 Matching arcs for noncrossing cycles

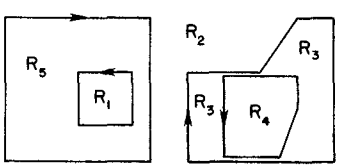

FIG 6 An example of cycles and the demarcated regions

Since the cycles are noncrossing and the graph is planar, the cycles partition the edges in U on the basis of being inside or outside certain cycles. A *region* is a maximal subset of U such that all edges in the subset are inside the same set of cycles and outside the remaining cycles. See Figure 6 for an example of several cycles and the regions they demarcate. Note that p cycles demarcate $p + 1$ regions.

Algorithm GETREGIONS

Input Output of GETCYCLES

Output A set of regions, each consisting of a set of edges and a set of indices indicating which cycles form the boundaries of the region

1 Mark each vertex not on a cycle as unexamined Mark each cycle as unexamined. Set q, the counter for the number of regions, to 0
2 While U is not empty
3 Increment q Extract an edge (u, v) from U and insert it into region(q) Call CHECKVERTEX((u, v), u) Call CHECKVERTEX((u, v), v)
4 While H is not empty
5 Extract a vertex v from H
6 If v is on a cycle(r) for some r,
7 Then
For each edge (v, w) incident on v in U with the proper handedness, extract the edge from U, insert it into region(q), and call CHECKVERTEX((v, w), w)
For each arc or oriented edge $\langle v, w\rangle$ on a cycle, say cycle(r), such that cycle(r) is unexamined and satisfies the handedness condition, mark cycle(r) as examined, insert all unexamined vertices of cycle(r) into H, associating the proper handedness with each vertex, mark each vertex as examined, and insert r into boundary(q)
8 Else
For each edge (v, w) incident on v in U, extract the edge from U, insert it into region(q), and call CHECKVERTEX((v, w), w)
9 Mark each cycle in boundary(q) as unexamined, and mark each vertex in each such cycle as unexamined

Procedure CHECKVERTEX is used to enter unexamined vertices into a holding set H If the vertex is on a cycle, then all the vertices on the cycle may be entered into H, if they have not already been entered. Edges incident on these vertices need be examined only if they have the proper handedness (either on the inside or the outside of the cycle).

Procedure CHECKVERTEX(edge e, vertex v)

1 If v is not on a cycle,
2 Then
If v is unexamined, then insert it into H and mark it as examined
3 Else
If v is unexamined, then determine the handedness of e with respect to the cycle, say cycle(r), insert v into H with appropriate handedness, and mark v as examined

Algorithms GETCYCLES and GETREGIONS partition M and U into a workable form. Algorithm HANDLEREGION contains the heart of SMALLCYCLES. The Chinese postman algorithm is performed for each region with its boundary cycles shrunk to single nodes, and portions of the cyles are duplicated to produce a set from which a tour can be constructed.

Algorithm HANDLEREGION

Input A region and the cycles that demarcate it

Output A multiset M'' of arcs and oriented edges, covering the region, such that for each vertex the indegree equals the outdegree

1 For each cycle forming a boundary of the region, map the vertices of that cycle to a single node representing the cycle Edges incident on vertices in a cycle will be incident on the corresponding node, and in the same order circularly about the node as they were circularly about the cycle

2 Perform the Chinese postman algorithm for the region
3 Call ASSIGNDIRECTION (which assigns direction to the postman edges, so that at any node the edges alternate in a circular direction around the node between incoming and outgoing)
4 Expand the nodes representing cycles back into cycles
5 Duplicate the portions of the cycles between an incoming postman edge and an outgoing postman edge, going around the cycle in the direction of the cycle arcs
6 If the additional cycle cost is more than half the length of the involved cycles, then reverse the direction on the Chinese postman edges, and replace the additional cycle portions with their complement in the cycle
7 Insert the postman edges, with proper orientation, and the cycle portions, into M''

Algorithm ASSIGNDIRECTION

Input A planar graph in which every vertex is of even degree

Output The same graph with directions assigned to each edge, so that the edges adjacent to a vertex alternate as to incoming and outgoing

1 Mark all vertices as unexamined
2 Choose an edge (u, v) from the graph Going in a circular direction around u, for every second edge (u, w) starting with (u, v), assign it direction from u to w and insert it into H Mark u as examined
3 While H is not empty,
4 Extract an edge (u, v) from H, with direction from u to v
5 If v is unexamined,
6 Then
Find the clockwise adjacent edge (v, w) with respect to (u, v) For every second edge (v, x) starting with (v, w), give the edge direction from v to x and insert it in H Mark v as examined

As an example for HANDLEREGION we consider region R_5 of Figure 6. The cycles bordering region R_5 are shown shrunk to nodes in Figure 7. The example is continued with the two choices for the duplication of cycle edges shown in Figure 8.

LEMMA 5. *Algorithm GETCYCLES produces a set of cycles that are noncrossing.*

PROOF. Select any arc and follow its clockwise adjacent successor repeatedly until an arc would be traversed a second time. Such an arc can be found, since the number of arcs is finite and each vertex with an incoming arc will have an outgoing arc. A cycle will thus be identified such that for every vertex on the cycle there will be no outgoing arc between the incoming and outgoing cycle arcs in a clockwise direction around the vertex Since the graph is planar, there can be no incoming arc between the two cycle arcs.

At each vertex the incoming and outgoing cycle arcs will satisfy the criterion in step 6 of GETCYCLES. Thus the cycle is one of the cycles identified by GETCYCLES. Further, no cycle can cross this cycle since there are no arcs, either incoming or outgoing, between the incoming and outgoing cycle arc in a clockwise direction around any vertex on the cycle. Now remove the arcs of the cycle from the graph. For each vertex the indegree will still equal the outdegree, and the remaining incoming and outgoing arcs will correspond to each other as before. The process of identifying a cycle on the reduced graph can be repeated until all cycles have been found, each of which will be noncrossing and a cycle identified by GETCYCLES. □

LEMMA 6. *Algorithm ASSIGNDIRECTION will assign direction to the edges of a planar graph in which every vertex has even degree, so that the edges adjacent to a vertex alternate as to incoming and outgoing.*

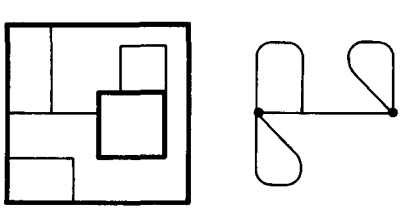

FIG 7 Shrinking cycles to nodes

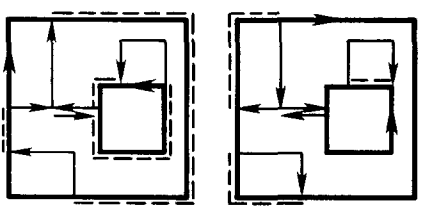

FIG 8 Two choices in duplicating portions of cycles

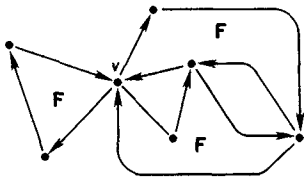

FIG 9 Faces and edge directions

PROOF. We first prove that an acceptable labeling of the edges does exist. We paraphrase a result from [1, Theorem 4-23]: If every vertex of a planar graph has even degree, then the faces of the graph will be two-colorable. Therefore the faces of the graph, which is input to Algorithm ASSIGNDIRECTION, will be two-colorable. Consider a two-coloring of the faces. Choose one of the face colors, and assign to the edges bordering all the faces of that color a clockwise direction with respect to the face. Each edge borders a face of each color, since the faces are two-colored. The edge will thus be assigned just one direction. Further, the faces meeting at a vertex v must alternate in color around the vertex v, so that the directions of edges incident on v will alternate around v. An example of this is shown in Figure 9.

It is easy to see that only one other acceptable edge labeling exists, since all directions could be reversed and the desired property would be preserved Algorithm ASSIGNDIRECTION is guaranteed to find one of the two acceptable edge labelings, since it labels edges in a fashion consistent with an acceptable labeling. That is, if an edge (v, w) is assigned direction from v to w, then only those edges incident on w that should be directed from w are directed from w. This is because we know that incoming and outgoing edges must alternate. □

In order to minimize the cycle portions that must be duplicated in SMALLCYCLES, each region should be handled separately. The next lemma shows that performing the Chinese postman algorithm on each region separately will not increase the cost of SMALLCYCLES.

LEMMA 7. *If the cycles are shrunk to nodes, then the cost of a Chinese postman algorithm for all edges will be equal to the total cost of Chinese postman algorithms run separately on edges in each region.*

PROOF. Let G' be the graph resulting from shrinking each cycle of G to a single node. Let G'' be the graph resulting from running the Chinese postman algorithm on G'. Since the graph is planar, each edge in a matching path used to produce G'' can be assigned to some region. Thus we can partition G'' into subgraphs $G''_1, G''_2, \ldots, G''_q$ corresponding to regions. We note that a node representing a shrunken cycle will be present in each (at most two) of the subgraphs for a region that the cycle bordered.

Now from the property of the Chinese postman algorithm, every vertex in G'' is of even degree. Thus the only vertices in a subgraph G''_i that could be of odd degree are those that correspond to a shrunken cycle and therefore are the result of splitting a node of even degree into two vertices of odd degree. We argue inductively that no such odd vertices exist. Consider the subgraph G''_j corresponding to an innermost region of G, which must be bounded by just one cycle. Now the total degree of any subgraph must be even, so that there cannot be a single vertex of odd degree. Thus the node corresponding to the boundary cycle must have been split into two vertices of even degree. By working outward from innermost regions, each node in G'' corresponding to a shrunken cycle can be shown to be split into vertices of even degree. Thus every vertex in every subgraph G''_i is of even degree.

Thus there exist matchings for each of the regions at total cost no greater than a minimum-cost postman matching for the whole graph G'. Similarly, given minimum-cost postman matchings for each of the regions, these need only be merged for a matching of equal cost for the whole graph G'. Thus the cost of a postman matching on the whole graph G' will equal the sum of the costs of the postman matchings on the regions of G'. □

Although our algorithm performs a Chinese postman algorithm for each region separately, we may thus, as a result of Lemma 7, view our algorithm as performing a Chinese postman algorithm on G'. We use this characterization in the next lemma.

LEMMA 8. *The cost of the cycles produced by Algorithm INOUTDEGREE plus the cost of the Chinese postman algorithm on the graph with the cycles shrunk to nodes is at most* C^*.

PROOF. Algorithm INOUTDEGREE produces a set M of arcs and oriented edges that can be used to form cycles, and a set U of undirected edges that are not used in the cycles. Let C'_M be the cost of additional copies of arcs and oriented edges in M. Algorithm INOUTDEGREE will minimize this cost. Let V_M be the set of vertices that are endpoints of arcs or oriented edges in M. Let $V_{M1}, \ldots, V_{Mk}$ be a partition of V_M such that each V_{Mi} is the set of vertices in a strongly connected component of the subgraph (V_M, M). We note that there is no 1–1 correspondence between cycles and V_{Mi}: If two cycles share a vertex, then all vertices from both cycles should be in the same V_{Mi}.

Now the Chinese postman algorithm is performed on the graph $G' = (V', U)$ that is derived from G by mapping all vertices in each set V_{Mi} into a single vertex v_i and removing all edges and arcs in M. Thus a minimum-cost matching U' is performed between vertices of odd degree from G', using edges in U. Let C'_U be the cost of edges in U', which are duplicates of edges in U. Thus the total cost of additional edges and arcs in $M \cup U \cup U'$ will be $C'_M + C'_U$.

Now assume that an optimal tour T^* exists with total cost of additional edges and arcs less than $C'_M + C'_U$. Let U^* be the multiset of those edges taken from T^* and disoriented that are elements of U. Let $U^{*\prime}$ be the set obtained from $U^* - U$ by deleting as many pairs of identical additional edges as possible. Thus $U^{*\prime}$ will turn out by construction not to contain multiple copies of any edge. Let $C^*_{U'}$ be the cost of edges in $U^{*\prime}$.

Since T^* is a tour, the degree of each vertex in T^* is even. If we then shrink each vertex set V_{Mi} in T^*, we see that the degree of each vertex in V' relative to U^* must also be even. From the manner in which $U^{*\prime}$ is constructed, we can see that the parity of each vertex in V' relative to $U^{*\prime} \cup U$ is the same as the parity relative to U^*, which is even. Thus we can conclude that $U^{*\prime}$ is a matching of odd vertices in G', and thus $C^*_{U'} \geq C'_U$.

Augment G to yield $G^{*\prime}$ by unioning $U^{*\prime}$ into the multiset E of edges. The edges and arcs in $G^{*\prime}$ are thus a subset of the edges and arcs in T^*, and the indegree and outdegree of any particular vertex in $G^{*\prime}$ need not be equal. Let $C^*_{M'}$ be the cost of edges and arcs in T^* minus the cost of edges and arcs in $G^{*\prime}$. Let C^+_M be the cost of additional arcs and edges added via a minimum-cost flow problem to make indegree equal to outdegree for each vertex in $G^{*\prime}$. We have $C^*_{M'} \geq C^+_M$, since indegree and outdegree are made equal in minimum-cost fashion. Next we demonstrate that $C^+_M \geq C'_M$.

If $C^+_M < C'_M$, then we show that we could achieve a minimum-cost way of making indegree and outdegree equal for G in less additional cost than C'_M. We recall that M is the set of arcs and oriented edges resulting from applying INOUTDEGREE to G. Let M^+ be the set of arcs and oriented edges resulting from applying INOUTDEGREE to $G^{*\prime}$. Our aim is to partition $M \cup M^+$ into two sets, M_0 and M_1, each of which satisfies the criteria that M satisfies. Insert one copy of each arc into both M_0 and M_1. For each edge in $U^{*\prime}$ such that it and a duplicate are both oriented in M^+, put one oriented copy in each of M_0 and M_1.

We complete the partition by creating a 0–1 circulation on $M \cup M^+$ as follows. Place upper and lower bounds and a flow of zero on each element of M_0, and upper and lower bounds and a flow of one on each element of M_1. Place an upper bound of one, a lower bound of zero, and a flow of one-half on each element of $(M \cup M^+) - (M_0 \cup M_1)$. Since indegree equals outdegree at each vertex in $M \cup M^+$, the above assignment of flows is a feasible circulation. By the integrality theorem [14] there exists a feasible 0–1 circulation. Edges and arcs with flow zero will be in M_0, and those with flow one will be in M_1. Since edges in $U^{*\prime}$ do not occur in M and are split up if they occur in pairs in M^+, we have two new sets that satisfy the criteria for G that M satisfies. However, since their combined

additional cost is $C_M^+ + C'_M$, and assuming $C_M^+ < C'_M$, one must be of additional cost less than C'_M. This is a contradiction of the fact that M is achieved in minimum-cost fashion. Thus $C_M^+ \geq C'_M$.

From considering the set U, we have $C_U^{*\prime} \geq C'_U$. From the preceding argument we have $C_M^{*\prime} \geq C_M^+ \geq C'_M$. Therefore we have $C_U^{*\prime} + C_M^{*\prime} \geq C'_U + C'_M$. This is a contradiction of the assumption that there was a tour T^* with cost of additional arcs and edges being less than $C'_U + C'_M$. □

LEMMA 9. *The cost of a tour generated by Algorithm SMALLCYCLES is no more than* $C_M + C^*$.

PROOF. We know from Lemma 7 that the total cost of the Chinese postman algorithm run on each of the regions with cycles shrunk to nodes will be no more than the cost of a Chinese postman algorithm run on the whole graph with the cycles shrunk to nodes. From Lemma 8 we know that the cost of the cycles plus the cost of the Chinese postman algorithm performed on the graph with the cycles shrunk to nodes will be no more than C^*.

When nodes are expanded back into cycles, portions of the cycles will be duplicated, and this cost will be no more than C_M. For each region the cost of duplicated cycles is no more than half the cost of the cycles that form its boundary. Since regions are defined to be maximal subsets, each cycle can be part of the boundary for at most two regions, one on the inside and one on the outside of it. Thus the total cost is at most $2 * \frac{1}{2}C_M$. Therefore the cost of the tour will be the total cost of the cycles, the postman edges, and the duplicate portions of cycles, which will equal $C_M + C^*$. □

We now specify the complete approximation algorithm for the planar mixed postman problem. The output of INOUTDEGREE will be used as input to both LARGECYCLES and SMALLCYCLES, and the better of the two results will be chosen.

Algorithm PLANARMIXED

Input A weighted, mixed graph G

Output. A tour that covers all edges and arcs in G

1. Call INOUTDEGREE
2. Call LARGECYCLES
3. Call SMALLCYCLES
4. Select the tour of smaller cost

THEOREM 4. *Algorithm PLANARMIXED produces a tour such that*

$$\hat{C}/C^* \leq \tfrac{3}{2}$$

and the bound is approachable. The time complexity of PLANARMIXED is the same as that of MIXED1.

PROOF. If $C_M > \frac{1}{2}C^*$, then consider the result from Lemma 3 concerning Algorithm LARGECYCLES:

$$\begin{aligned}\hat{C}/C^* &\leq (2C^* - C_M)/C^* \\ &< (2C^* - \tfrac{1}{2}C^*)/C^* \\ &= \tfrac{3}{2}.\end{aligned}$$

If $C_M \leq \frac{1}{2}C^*$, then consider the result from Lemma 9 concerning Algorithm SMALLCYCLES:

$$\begin{aligned}\hat{C}/C^* &\leq (C_M + C^*)/C^* \\ &\leq (\tfrac{1}{2}C^* + C^*)/C^* \\ &= \tfrac{3}{2}.\end{aligned}$$

To see that the bound is approachable, consider the graph in Figure 10 (a) An optimum solution is shown in Figure 10 (b). It consists of doubling two of the arcs of cost ϵ and will

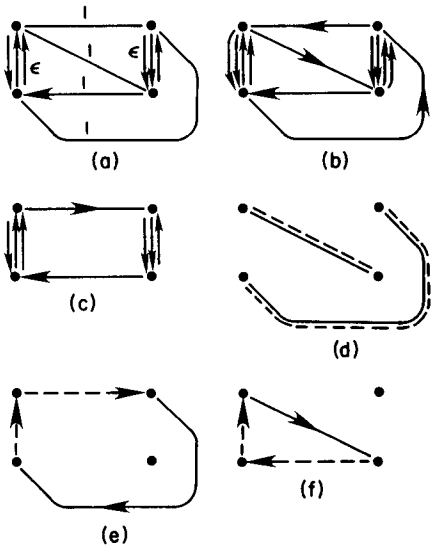

FIG 10 A worst-case example for Algorithm PLANARMIXED

cost 4 + 8ϵ. The cycles that could be generated by INOUTDEGREE are shown in Figure 10 (c). The edges that must be duplicated in LARGECYCLES are shown in Figure 10 (d). The cost of LARGECYCLES will be 6 + 6ϵ. In Figure 10 (e) and 10 (f), two of the regions of SMALLCYCLES are shown with the bordering cycles and the portion of the cycle that must be duplicated. The cost of SMALLCYCLES will be 6 + 6ϵ. Thus the cost of PLANARMIXED will be 6 + 6ϵ. As ϵ goes to zero, the worst-case bound of $\frac{3}{2}$ is realized.

Algorithm SMALLCYCLES is dominated by step 2 of HANDLEREGIONS, which will require $O(|V|^3)$ [5, 9]. □

7. *Conclusion*

We have bounded the heuristic of Edmonds and Johnson for the mixed postman problem and have developed an equally good (in the worst case) heuristic. A mixed strategy approach has been employed to yield a better bound for the mixed postman for both general graphs and planar graphs. The mixed strategy appears to be a natural approach for the mixed postman problems, since two conditions must be satisfied in augmenting the graph to yield the basis of a tour.

ACKNOWLEDGMENTS. I would like to thank my doctoral advisor Prof. Matthew S. Hecht for his guidance during the preparation of this paper. I would also like to thank the referees for many helpful suggestions.

REFERENCES

1 BUSACKER, R G, AND SAATY, T L *Finite Graphs and Networks* McGraw-Hill, New York, 1965
2 CHRISTOFIDES, N Worst-case analysis of a new heuristic for the traveling salesman problem Manage Sci Res Rep No 388, Carnegie-Mellon U, Pittsburgh, Pa, 1976
3 COOK S A The complexity of theorem-proving procedures Proc Third Annual ACM Symp Theory of Comptng, Shaker Heights, Ohio, May 1971, pp 151–158
4 EDMONDS, J The Chinese postman problem *Oper Res 13*, Suppl 1 (1965), B73–B77
5 EDMONDS, J, AND JOHNSON, E L Matching, Euler tours and the Chinese postman *Math Programming 5* (1973), 88–124
6 FLOYD, R Algorithm 97 Shortest path *Comm ACM 5*, 6 (June 1962), 345
7 FORD, L R, AND FULKERSON, D R *Flows in Networks* Princeton U Press, Princeton, N J, 1962
8 FREDERICKSON, G N, HECHT, M S, AND KIM, C E Approximation algorithms for some routing problems *Siam J Compting 7* (1978), 178–193
9 GABOW, H, AND LAWLER, E L An efficient implementation of Edmonds' algorithm for maximum weight

matching on graphs TR CU-CS-075-75, Dept Comptr Sci, U of Colorado, Boulder, Colo, 1975
10 GAREY, M R, AND JOHNSON, D.S. Approximation algorithms for combinatorial problems. An annotated bibliography In *Algorithms and Complexity Recent Results and New Directions*, J.F Traub, Ed, Academic Press, New York, 1976
11 IBARRA, O.H., AND KIM, C E Fast approximation algorithms for the knapsack and sum of subset problems *J ACM 22*, 4 (Oct. 1975), 463–468.
12 JOHNSON, D S Approximation algorithms for combinatorial problems *J Comptr. Syst Sci. 9* (1974), 256–278
13 KARP, R M Reducibility among combinatorial problems In *Complexity of Computer Computations*, R E Miller and J W Thatcher, Eds, Plenum Press, New York, 1972, pp 85–104
14 LAWLER, E L *Combinatorial Optimizations Networks and Matroids* Holt, Rinehart, and Winston, New York, 1976
15 LENSTRA, J,K, AND RINNOOY KAN, A H G On general routing problems *Networks 6* (1976), 273–280
16 MEI-KO, K Graphic programming using odd or even points *Chinese Mathematics 1* (1962), 237–277
17 ORLOFF, C S A fundamental problem in vehicle routing *Networks 4* (1974), 35–64
18 ORLOFF, C S On general routing problems Comments *Networks 6* (1976), 281–284
19 PAPADIMITRIOU, C H On the complexity of edge traversing *J ACM 23*, 3 (July 1976), 544–554
20 ROSENKRANTZ, D J, STEARNS, R E, AND LEWIS, P M An analysis of several heuristics for the traveling salesman problem *SIAM J Compting 6* (1977), 115–124
21 SAHNI, S K Approximate algorithms for the 0/1 knapsack problem *J ACM 22*, 1 (Jan 1975), 115–124

RECEIVED APRIL 1977, REVISED DECEMBER 1978