Mathematical Programming 5 (1973) 88–124. North-Holland Publishing Company

MATCHING, EULER TOURS AND THE CHINESE POSTMAN

Jack EDMONDS
University of Waterloo, Waterloo, Ontario, Canada

and

Ellis L. JOHNSON
IBM Watson Research Center, Yorktown Heights, New York, U.S.A.

Received 20 May 1972
Revised manuscript received 3 April 1973

The solution of the Chinese postman problem using matching theory is given. The convex hull of integer solutions is described as a linear programming polyhedron. This polyhedron is used to show that a good algorithm gives an optimum solution. The algorithm is a specialization of the more general *b*-matching blossom algorithm. Algorithms for finding Euler tours and related problems are also discussed.

1. Introduction

One of the oldest problems in graph theory is that of finding an Euler tour in a connected graph [9]. The problem is to find a way of traversing every edge exactly once in a tour of the graph. Necessary and sufficient conditions for the existence of such a tour, an Euler tour, are simple: each node must be incident to an even number of edges.

We shall give a related optimization problem, the Chinese postman problem, and its solution using matching theory. Matching theory provides not only a good algorithm for the Chinese postman problem but also a description of the polyhedron of solutions. In Section 4, a variant of the matching algorithm is given for the Chinese postman problem. This algorithm does not actually give us a postman tour but instead a graph in which an Euler tour exists. We discuss in Section 5 the problem of finding an Euler tour. Finally, Sections 6 and 7 discuss similar problems except that some edges are directed.

2. The Chinese postman problem

The Chinese postman problem [6, 14] is to find the shortest tour such that each edge is traversed at least once. Thus, it is the problem faced by a postman who must deliver mail along each edge of a graph and return to his starting point.

To be more precise, define a *graph* G to be a finite set N of *nodes* and a finite set E of *edges* such that each edge *meets* two nodes. We allow more than one edge to meet the same pair of nodes. A *tour* in G is a sequence

$$(n_1, e_1, n_2, e_2, \ldots, n_l, e_l, n_{l+1} = n_1)$$

of nodes n_i and edges e_i such that e_i meets the distinct nodes n_i and n_{i+1}. A *simple tour* of G is a circuit which contains each edge $e \in E$ at most once. A *postman tour* of G is a circuit which contains every edge e at least once. An *Euler tour* of G is a tour which contains every edge exactly once. A *path* is a sequence of nodes and edges, like a circuit, except that n_{l+1} is not required to be equal to n_1. An *edge-simple path* is a path which contains each edge at most once. A *connected* graph G has a path between every pair of nodes. The length c_e of an edge e is assumed to be a non-negative number, and the length of a tour $(n_1, e_1, n_2, \ldots$ $\ldots, n_l, e_l, n_1)$ is $\Sigma_{i=1}^{l} c_{e_i}$. The Chinese postman problem is to find the minimum length postman tour of a connected graph G.

If there is an Euler tour in the graph, then it solves the Chinese postman problem. Thus, whenever every node of a connected graph is incident to an even number of edges, the Chinese postman problem reduces to simply finding an Euler tour, which is known to exist in such a graph.

On the other hand, given any postman tour of G, every edge e is in the tour at least once, but perhaps more than once. Let $1 + x_e$ be the number of times edge e is in the tour. Let G' be formed from G by putting x_e additional copies of edge e in G'. That is, where G had one copy of edge e, G' has $1 + x_e$ copies of edge e. Then the postman tour of G becomes an Euler tour of G'. In the graph G', every node is incident to an even number of edges.

Finding the numbers x_e of an optimum postman tour is equivalent to the problem of finding an integer $x_e \geq 0$ for every edge e of G such that $\Sigma c_e x_e$ is minimized subject to $\Sigma_e(1 + x_e) \equiv 0 \pmod 2$, where the sum is over the edges e meeting node i, for each node i of G. When such x_e are found, then the graph G' having $1 + x_e$ copies of edge e has an Euler tour which is the optimum postman tour in the original graph G.

The Chinese postman problem can, thus, be separated onto two parts: finding optimum x_e to the above problem, and then finding an Euler tour, which is known to exist, in the resulting graph G'. The more difficult part, finding optimum x_e, can be solved by the matching algorithm.

3. Matching and parity constraints

In order to describe the Chinese postman problem as a matching problem, let the *node-edge incidence matrix* (a_{ne}), $n \in N$ and $e \in E$, be defined by

$$a_{ne} = \begin{cases} 1 & \text{if edge } e \text{ meets node } n\,, \\ 0 & \text{otherwise}\,. \end{cases}$$

The problem just described is to find integers $x_e \geq 0$, $e \in E$, such that

$$\sum_{e \in E} a_{ne}\,(1 + x_e) \equiv 0 \pmod 2$$

and minimizing $\Sigma\, c_e x_e$. The above congruence can be written as

$$\sum_{e \in E} a_{ne}\, x_e \equiv \sum_{e \in E} a_{ne} \pmod 2\,.$$

The sum $\Sigma_e a_{ne}$ is the *degree* of node n; that is, the number of edges meeting node n. Let b_n be zero or one given by

$$b_n \equiv \sum_{e \in E} a_{ne} \pmod 2\,.$$

Thus, b_n is zero when the degree of n is even and is one when the degree of n is odd. Call node n an *odd node* when $b_n = 1$ and an *even node* when $b_n = 0$.

The Chinese postman problem can be solved if we can find x_e, $e \in E$, such that:

$$x_e \text{ is integer}, \quad e \in E; \tag{3.1}$$
$$w_n \text{ is integer}, \quad n \in N; \tag{3.1'}$$
$$x_e \geq 0, \quad e \in E; \tag{3.2}$$
$$w_n \geq 0, \quad n \in N; \tag{3.2'}$$

$$\Sigma_{e \in E} a_{ne} x_e - 2w_n = b_n, \quad n \in N; \tag{3.3}$$

$$z = \Sigma c_e x_e \text{ is minimized.} \tag{3.4}$$

The integer x_e represents the number of extra times the edge e is traversed. In terms of the postman, x_e is the number of times he must traverse an edge without delivering mail. The lengths c_e are assumed to be non-negative so that $x_e = 0$ for all $e \in E$ and $w_n = 0$ for all $n \in N$ is an optimum solution whenever all $b_n = 0$. This observation reflects the fact that when every node has even degree, there is an Euler tour.

The problem (3.1)–(3.4) is a special case of the general matching problem [7]. Each variable w_n appears in only one equation (3.3), so the extension of the coefficient matrix to include the coefficients of w_n does have $\Sigma_n |a_{ne}| \leq 2$ as required [7, p. 89]. The variables w_n can be thought of as adjoining loops to the graph at each node, where a *loop* is an edge with two ends meeting the same node. The edges $e \in E$ are assumed to meet two different nodes, although allowing loops in E does not change the problem since their use would not effect the congruence $\Sigma a_{ne} x_e \equiv b_n \pmod 2$. The matching polyhedron theorem and the matching algorithm thus apply to this problem.

The matching polyhedron description says that the convex hull of integer solutions x_e, $e \in E$, w_n, $n \in N$, is the same as the polyhedron of linear restrictions (3.2), (3.2′), (3.3), and some additional inequalities, called blossom inequalities. In order to describe the blossom inequalities for this special case, let us say that an edge e *meets* a set S of nodes whenever e meets one node in S and one node not in S. Define an *odd set* S of nodes to be a set of nodes containing an odd number of odd nodes (and any number of even nodes). Then the blossom inequalities become

$$\sum \{x_e : \ e \text{ meets } S\} \geq 1 \ \text{ for } S \text{ an odd set}. \tag{3.5}$$

The Chinese postman polyhedron is the set of solutions to (3.2), (3.2′), (3.3) and (3.5). This polyhedron is equal to the convex hull of integer points satisfying (3.1), (3.1′), (3.2), (3.2′), (3.3).

Matching theory also provides us with a good algorithm for solving this problem. In fact, the polyhedral description is proven by showing that the matching algorithm will terminate at a solution to (3.1), (3.1′), (3.2), (3.2′), (3.3) which is optimum to the linear program of minimizing (3.4) subject to (3.2), (3.2′), (3.3), (3.5).

Another way to use matching algorithms for this parity problem is to

reduce the problem to an equivalent 1-matching problem [4, 5]. Begin by finding a shortest path between every pair of odd nodes. There are algorithms available to find all such shortest paths and which require of the order of $|N|^3$ operations [12], where $|N|$ is the number of nodes in the graph G. Now form another graph G_p whose nodes are the odd nodes of G and with one edge of G_p between every pair of nodes of G_p. Let the length of an edge in G_p be the length of the shortest path between the two odd nodes which the edge meets. Let a particular shortest path between two odd nodes be associated with the edge in G_p. Find an optimum 1-matching of G_p; that is, a set M of edges of G_p such that every node is met by exactly one edge of M and the sum of the lengths in M is minimum. To find the solution x_e to (3.1), (3.2) and (3.3), for every edge m of G_p in M let $x_e = 1$ for every edge e of G in the shortest path corresponding to that edge m of M.

To show that the above procedure does solve (3.1), (3.2) and (3.3), we first show that no edge e of G will be in two shortest paths corresponding to matching edges of G_p. Suppose the contrary: let edge e of G be in the shortest path from i_1 to i_2 and in the shortest path from j_1 to j_2 and let both paths correspond to matching edges of G_p. Both paths can be assumed to be edge-simple because they are shortest paths in a graph having non-negative lengths on every edge. Removing e from both paths leaves edges of G forming a path from i_1 to j_1 (or j_2) and a path from i_2 to j_2 (or j_1). The sum of the lengths of these two paths is less than or equal to the sum of the lengths of the two paths, from i_1 to i_2 and from j_1 to j_2, corresponding to matching edges. Hence, the sum of the lengths of edges $(i_1, j_1), (i_2, j_2)$ in G_p is less than or equal to the sum of the lengths of edges $(i_1, i_2), (j_1, j_2)$. Therefore, edges $(i_1, j_1), (i_2, j_2)$ can replace edges $(i_1, i_2), (j_1, j_2)$ in the matching M of G_p to give at least as good a matching M' and such that the shortest paths in G corresponding to matching edges of G_p contain edge e two fewer times than previously while containing every other edge of G the same number of times. In this way, we can change the matching so that no edge e of G is in more than one shortest path corresponding to matching edges. We can, then, assign $x_e = 1$ to every edge e of G in a shortest path corresponding to a matching edge.

The resulting x_e, $e \in E$, obviously satisfies (3.1) and (3.2). The remaining question is whether non-negative integer w_n can be found so that (3.3) is satisfied. Such w_n can be found provided the sum of the x_e over e meeting node n is even for even nodes n and odd for odd nodes n. This sum over x_e in a shortest path corresponding to any one matching

edge is even for all nodes except for the two odd nodes at the ends of the path. Since every odd node is an end point of exactly one shortest path corresponding to a matching edge, the result follows.

We have shown that from an optimum matching M in G_p, a solution x_e to (3.1), (3.2) and (3.3) can be found. That solution must be shown to be optimum to the problem of minimizing z subject to (3.1), (3.2) and (3.3). Consider an optimum solution. It can be assumed to have x_e either zero or one, because otherwise x_e could be reduced by two, and, by $c_e \geq 0$, the cost would decrease or remain the same. It is easy to show that the edges for which $x_e = 1$ can be partitioned into edge-simple paths connecting pairs of odd nodes. The reason is that an odd number of edges e having $x_e = 1$ meet odd nodes, and an even number meet even nodes. We can, thus, begin at any odd node and begin following edges, arbitrarily, until another odd node is met. Removing those edges traversed results in edges e with $x_e = 1$ for which an even number meet even nodes and an odd number meet odd nodes except for the two just connected by a path. Begin again at any remaining odd node and follow a path to another odd node. Repeating eventually provides paths between pairs of odd nodes. The only remaining edges e having $x_e = 1$, after every odd node is connected by some path to some other odd node, have even degree at each node so can be decomposed into tours, if any such edges remain. These $x_e = 1$ can all be reduced to zero without violating (3.1), (3.2) and (3.3) and without increasing z by $c_e \geq 0$. Therefore, an optimum solution to (3.1), (3.2) and (3.3) can be represented as paths connecting pairs of odd nodes. Since our optimum matching provided the minimum z for such a collection of paths, the matching gives an optimum solution to (3.1), (3.2) and (3.3).

Returning to the Chinese postman polyhedron (3.2), (3.2′), (3.3) and (3.5), we will sharpen that result by giving the polyhedron in terms of x_e, $e \in E$, only without the w_n. Consider the set of x_e, $e \in E$, satisfying (3.1), (3.2) and

$$\sum_{e \in E} a_{ne} x_e \equiv b_n \pmod 2, \quad n \in N. \tag{3.6}$$

We wish to describe the convex hull of (3.1), (3.2) and (3.6) as a polyhedron. In fact, that convex hull is equal to the polyhedron of solutions to (3.2) and (3.5). To prove this result, notice that in (3.3) non-negativity of w_n is automatically satisfied whenever all the rest of (3.1), (3.1′), (3.2) and (3.3) are satisfied. The reason is that

$$2w_n = \sum_{e \in E} a_{ne} x_e - b_n ,$$

where b_n is 0 or 1, so $2w_n$ takes on values $\{0, 1, ...\}$, or $\{-1, 0, 1, ...\}$. If w_n is integer, then $2w_n$ cannot be -1, so w_n must be non-negative. Therefore, (3.1), (3.1'), (3.2) and (3.3) describe the same set of integer points as (3.1), (3.1'), (3.2), (3.2') and (3.3). In other words, we can drop the restrictions $w_n \geq 0$, $n \in N$, in the inequalities (3.2), (3.3) and (3.5) without changing the polyhedra.

Considering (3.3), (3.5) and $x_e \geq 0$, $e \in E$, as a system of linear restrictions (3.3), the variables w_n can be dropped to give an equivalent, reduced system, since any $x_e \geq 0$ satisfying (3.5), (3.3) can be used to determine w_n, and the resulting x_e, w_n satisfy (3.3), (3.5), and $x_e \geq 0$. Therefore, the convex hull of solutions to (3.1), (3.2) and (3.6) is the polyhedron given by $x_e \geq 0$, $e \in E$,

$$\sum \{x_e : e \text{ meets } S\} \geq 1, \qquad S \text{ an odd set.}$$

The problem of minimizing $z = \Sigma c_e x_e$ over all x_e satisfying (3.1), (3.2) and (3.6) is the same as minimizing z over the above polyhedron, that is, (3.2) and (3.5). This latter minimization problem is a linear program and has a linear programming *dual problem* involving variables y_s for each odd set S. The dual constraints are

$$y_s \geq 0, \quad S \text{ an odd set,} \tag{3.7}$$

$$\sum \{y_s : S : e \text{ meets } S\} \leq c_e, \qquad e \in E, \tag{3.8}$$

and the objective function is to maximize

$$\sum \{y_s : \text{ odd sets } S\} = v. \tag{3.9}$$

The weak form of the linear programming duality theorem says that v given by (3.9) is less than or equal to $z = \Sigma c_e x_e$ subject to (3.2) and (3.5). This result is easily proven by multiplying each inequality (3.5) by $y_s \geq 0$ and subtracting from $z = \Sigma c_e x_e$ to give

$$\sum_{e \in E} \left(c_e - \sum \{y_s : S : e \text{ meets } S\} \right) x_e \leq z - \sum_S y_s. \tag{3.10}$$

By (3.8), each coefficient of x_e in (3.10) is non-negative, so $z \geq \Sigma_S y_s = v$ by $x_e \geq 0$.

In addition to showing $z \geq v$, the above argument can be used to derive sufficient conditions for $z = v$. Any pair $(x_e, e \in E)$, $(y_S$, odd sets $S)$ for which $z = v$ will be optimum to their respective linear programs. In order for $z = v$, from (3.10) it is clear that

$$x_e > 0 \text{ implies } \sum \{y_S\colon\ S\colon e \text{ meets } S\} = c_e \tag{3.11}$$

is needed. However, $z = v$ still may not hold if the inequality in (3.10) is not equality. That inequality came from multiplying (3.5) by y_S and subtracting from $\Sigma c_e\, x_e = z$. The inequality will hold with equality provided

$$y_S > 0 \text{ implies } \sum \{x_e\colon e \text{ meets } S\} = 1. \tag{3.12}$$

If both (3.11) and (3.12) hold, then $(x_e, e \in E)$ is an optimum solution to the linear program having constraints (3.2) and (3.5), and $(y_S$, odd sets $S)$ is an optimum solution to the dual problem (3.7), (3.8), (3.9).

The algorithm in Section 4, which is a variant of Edmonds' blossom algorithm for the matching problem, produces such a pair $(x_e, e \in E)$ and $(y_S$, odd sets $S)$ for which x_e is integer-valued and satisfies the parity constraint (3.6). In the same way, the blossom algorithm finds a matching for the general matching problem and a dual solution which are optimum pairs to the linear program over the matching polyhedron and its dual linear program. Thus, the algorithm proves the polyhedron result, which in turn proves optimality of the solution generated by the algorithm. For this special case, we will recreate that argument.

Solutions $(x_e, e \in E)$ to (3.1), (3.2) and (3.6) can be shown to satisfy (3.5), for a given odd set S, by adding (mod 2) the congruences (3.6) over $n \in S$ to give

$$\sum \{x_e\colon e \text{ meets } S\} \equiv 1 \pmod 2. \tag{3.13}$$

The right-hand side above is 1 because S is an odd set. We use the fact (3.1) that x_e is an integer in adding these congruences. By (3.2), nonnegativity of x_e, the sum of x_e over e meeting S must be 1, 3, 5, ..., and hence (3.5) follows.

Therefore, the convex polyhedron of solutions to (3.2) and (3.5) contains all of the $(x_e, e \in E)$ satisfying (3.1), (3.2) and (3.6). Hence, it contains the convex hull of the $(x_e, e \in E)$ satisfying (3.1), (3.2) and (3.6). We show that these two closed, convex sets are equal by showing

that the minimum of any linear objective function $z = \Sigma c_e x_e$ over the polyhedron (3.2) and (3.5) is achieved by a point $(x_e, e \in E)$ which also satisfies (3.1) and (3.6). Since our two convex sets are unbounded, we must also consider objective functions which can be made arbitrarily small. We must show that any linear objective function which can be made arbitrarily small using points $(x_e, e \in E)$ in (3.2) and (3.5) can also be made arbitrarily small using points $(x_e, e \in E)$ in the convex hull of (3.1), (3.2) and (3.6).

The algorithm will produce an optimum $(x_e, e \in E)$ over (3.2) and (3.5) for any objective function $z = \Sigma c_e x_e$ for which $c_e \geq 0$. If any $c_e < 0$, then z can be made arbitrarily small by increasing that x_e, beginning from any point in (3.2) and (3.5). For any point satisfying (3.2) and (3.5), increasing any component x_e will not cause (3.2) or (3.5) to be violated. However, this objective function can also be made arbitrarily small using solutions to (3.1), (3.2) and (3.6). For any solution to (3.1), (3.2) and (3.6), any component x_e can be increased by a positive even integer without violating (3.1), (3.2) or (3.6). So if any $c_e < 0$, then that x_e can be made arbitrarily large by even integer increases in x_e. Thereby, z can be made arbitrarily small.

The optimum $(x_e, e \in E)$ to the linear program of minimizing $z = \Sigma c_e x_e$ over (3.2) and (3.5), where $c_e \geq 0$, will satisfy (3.1) and (3.6) as well. The polyhedron of solutions to (3.2) and (3.5) is therefore equal to the convex hull of solutions to (3.1), (3.2) and (3.6) becuase they have the same set of supporting hyperplanes

$$\sum c_e x_e \geq z^*, \qquad c_e \geq 0, \tag{3.19}$$

where z^* is the common minimum value of $z = \Sigma c_e x_e$ subject to (3.2) and (3.5) or (3.1), (3.2) and (3.6).

4. The blossom algorithm

The algorithm for solving the parity constraint part of the Chinese postman problem described in Section 3 is first stated. Then we show that the steps can be executed, and the resulting solutions x_e and y_s satisfy (3.11) and (3.12). The algorithm given here is a variant of the matching algorithm [4, 8] applied directly to the graph G without introducing loops corresponding to W_n.

The values for x_e will be zero or one. If $x_e = 1$, the edge e will be re-

ferred to as a *matching edge*. An edge e which is not a matching edge, that is $x_e = 0$, is called a *non-matching edge*. We will have nodes which are either *pseudonodes* or *original nodes*. A pseudonode p corresponds to an odd set S and will have a dual variable y_p (or y_S) associated with it. The edges meeting p are the same as the edges meeting S. A single odd node of G can be made into a pseudonode. The original nodes are nodes of G.

At any given iteration of the algorithm, we will have a *surface graph* G_s made up of original nodes and pseudonodes such that the odd sets corresponding to pseudonodes of G_s are pair-wise disjoint and do not include any of the original nodes of G_s. Every original node of G will either be in G_s or will be in exactly one odd set corresponding to a pseudonode of G_s. The edges of G_s are the edges of G which do not meet two nodes in the same odd set corresponding to some pseudonode of G_s. In G_s, every matching edge will meet at least one pseudonode, and every pseudonode will be incident to at most one matching edge. Any pseudonode incident to no matching edge is called a *deficient pseudonode*.

In G_s, a *planted forest* F will be grown. Initially, $G_s = G$ except that odd nodes should be changed to deficient pseudonodes with the same incident edges. That is, odd (original) nodes are not in G_s, but for each one put a deficient pseudonode, with the same incident edges, in G_s. The initial planted forest will be those pseudonodes. In general, F (which need not be spanning) will consist of disjoint trees each containing one deficient pseudonode. The other nodes of F will be pseudonodes, and they will be alternately *outer* and *inner* nodes of F. The deficient pseudonodes will be outer nodes, and the other pseudonodes will be alternately outer and inner as illustrated in Fig. 4.1. Pseudonodes will be drawn as squares, and matching edges are indicated as wiggly lines. Outer nodes are indicated by a + and inner nodes by a −. Each

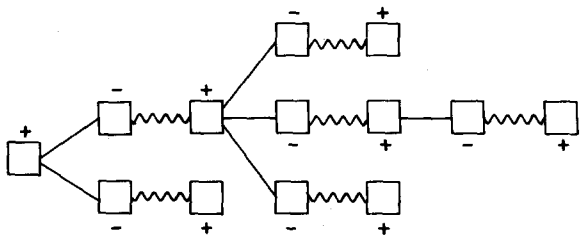

Fig. 4.1.

inner node is incident to exactly two edges, one of which is a matching edge. The outer nodes are incident to any number of non-matching edges and to one matching edge, except for the deficient pseudonode, which meets no matching edge. The deficient node is called the *root* of the planted tree.

Having designated odd nodes as deficient pseudonodes, the algorithm will now be stated. Each edge has associated with it a cost c_e. Let the *revised cost* c'_e initially be equal to c_e. Each node n of G_s will have three associated numbers: d_n^+, d_n^-, y_n. Initially, let $d_n^+ = +\infty$ and $y_n = 0$ except that $d_n^+ = 0$ if n is a deficient pseudonode. Begin with a planted forest consisting of the isolated deficient pseudonodes which are outer nodes of the planted forest. For every n in G_s, let

$$d_n^- = \min\{c'_e: \quad e \text{ meets an outer node } (\neq n) \text{ and } e \text{ meets } n\}.$$

Let k_n be e giving the minimum above. If there is no such e, then $d_n^- = +\infty$ and $k_n = 0$.

Step 1. Find the minimum over all $n \in G_s$ of:
(α) d_n^- for n not in any planted tree;
(β) $\frac{1}{2}(d_n^+ + d_n^-)$ for n an outer node of a planted tree;
(γ) $y_n + d_n^-$ for n an inner node of a planted tree.
Let d^* represent the minimum value given by node i of G_s. Go to (A), (B) or (C) depending on whether i giving the minimum was case (α), (β) or (γ), respectively.

(A)(1) If i is a pseudonode, then it becomes an inner pseudonode and is adjoined to a planted tree along with the edge k_i. The matching edge m meeting i is also adjoined to the planted tree. Let j be the other node incident to edge m.

(1a) If j is a pseudonode, then let j be an outer node, set d_j^+ to d^*, scan node j, and return to the beginning of Step 1. To *scan* node j means to look at every non-matching edge e of G_s and compare d_n^- to $d_j^+ + c'_e$, where n is the other node of G_s incident to edge e. If $d_j^+ + c'_e < d_n^-$, then let $k_n = e$ and $d_n^- = d_j^+ + c'_e$.

(1b) If j is an original node, then form a *blossom* B of node j and all matching edges, together with incident nodes, meeting node j. We *shrink* the blossom to form a new pseudonode p, and replace G_s by G'_s, where the edges of G'_s are the edges of G_s which do not meet two nodes of B and the nodes of G'_s are p and the nodes of G_s except for nodes of B. Edges of G_s which meet exactly one node of B meet p in G'_s.

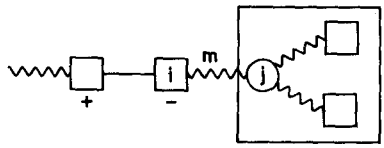

Fig. 4.2.

Pseudonode p becomes an outer node, d_p^+ is set equal to d^*, d_p^- is set to $+\infty$, and $y_p = 0$. Now scan (as in (1a)) the pseudonode p. Return to the beginning of Step 1.

(2) If i is an original node, let $k = k_i$, and let node n be the other node incident to k. Form a *blossom* B of node n, edge k, node i, and all matching edges meeting node i, together with incident nodes to such matching edges. As in (1b) replace G_s by G_s/B by shrinking B to form a new pseudonode p.

Pseudonode p is an outer node, d_p^+ is d^*, and $y_p = 0$. If pseudonode n was deficient, then so is pseudonode p. For each edge e of G (whether in G_s or not) meeting node n, subtract $d^* - d_n^+$ from the reduced cost c'_e. Change y_n to $y_n + d^* - d_n^+$. The number $d^* - d_n^+$ is equal to c_k, where $k = k_i$. Now scan the pseudonode p and return to the beginning of Step 1.

(B) Let j be the other node incident to edge k_i. If i and j are in different planted trees, go to Step 2. Otherwise, i and j are in the same planted tree, and edge k_i forms a circuit in that planted tree.

Nodes i and j are both outer nodes, so they both have alternating

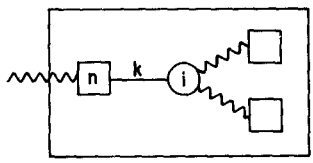

Fig. 4.3.

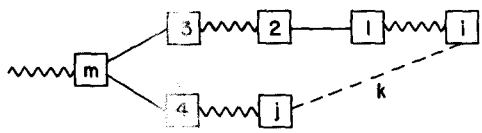

Fig. 4.4.

paths to the root of the planted trees. These paths meet at some outer node m as illustrated in Fig. 4.4. The circuit formed must have an odd number of edges.

Form a blossom B of the nodes and edges in the odd circuit and shrink the blossom to form a new pseudonode p. As in (A)(1b), replace G_s by G_s/B.

Pseudonode p is an outer node, d_p^+ is d^*, and $y_p = 0$. If pseudonode m is a deficient pseudonode, then so is pseudonode p. For each node n of B, that is, of the odd circuit, y_n will be changed and all incident edges e will have c'_e changed. A numerical example is given in Fig. 4.5. For each node n of B, replace y_n by $y_n + \Delta_n$, where $\Delta_n = d^* - d_n^+$ for n an outer node and $\Delta_n = -d^* + d_n^-$ for n an inner node (of the planted tree before shrinking B). Subtract Δ_n from every c'_e for e meeting node n. Scan node p (as in (A)(1a)) and return to the beginning of Step 1.

(C) This step is reached with an inner pseudonode i for which $d^* = y_i + d_i^-$.

(1) Begin by adding y_i to every edge e meeting pseudonode i and changing y_i to zero.

(2) *Expand* pseudonode i, that is, recover the blossom B which was shrunk to form pseudonode i and, also, the edges not in B which meet two nodes of B. The nodes of B may be pseudonodes, but they are not expanded. Let G_1 be defined by $G_s = G_1/B$, and replace G_s by G_1. Thus, delete node i from G_s and adjoin the nodes and edges in the expansion of i to G_s.

(3) Some matching and non-matching edges may need to be swapped in order to keep one matching edge incident to every pseudonode of B. This change is exactly as in Step 3 when pseudonodes are finally expanded. However, none of the pseudonodes of B are expanded here, only the pseudonode i is expanded to recover B.

(4) At this point, the planted tree is grown to include as much of B

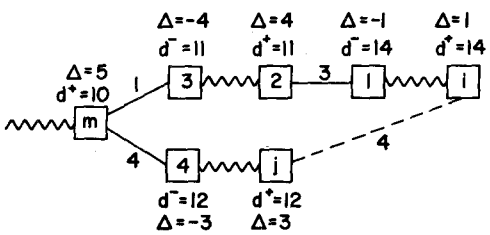

Fig. 4.5.

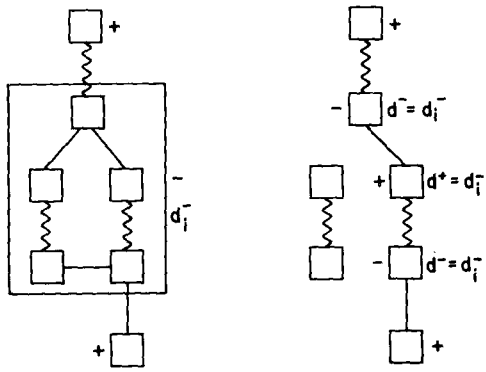

Fig. 4.6.

as possible. This part of (C) is much like (A) and will be considered in detail for the three different types of blossoms. In every case, there is a non-matching edge down toward the root from pseudonode i and a matching edge up from the root meeting i. Both edges meet outer nodes.

(a) The blossom may consist of an odd circuit as illustrated in Fig. 4.6. In this case, there is always an alternating path from the non-matching edge to the matching edge. This path becomes part of the planted tree with the nodes alternately inner and outer, d_n^+ for outer nodes n equal to d_i^-, and d_n^- for inner nodes n equal d_i^-, as well. These new outer nodes should be scanned as in (A)(1a).

The remainder of B (the part of the circuit not in the alternating path) consists of matching edges meeting two pseudonodes. For all of these nodes, say node n, let

$$d_n^- = \min\{d_i^+ + c'_e: \quad e \text{ meets an outer node } i \text{ and } e \text{ meets } n\}.$$

Let k_n be the e giving this minimum. If there is no such e, let $d_n^- = +\infty$ and $k_n = 0$. This part of (C) is similar to the initialization of d_n^- before Step 1 started except that here we use $d_i^+ + c'_e$ instead of c_e (these $d_i^+ = 0$ and $c'_e = c_e$). Return to the beginning of Step 1.

(b) The second type of blossom B has one original node and all matching edges as shown in Fig. 4.7. In this case, immediately reform the two outer nodes and the nodes and edges of B into a new outer node as in (A)(2). Here, for each of the two outer nodes n, subtract $d^* - d_n$ from c'_e for every incident edge e and add $d^* - d_n$ to y_n. The

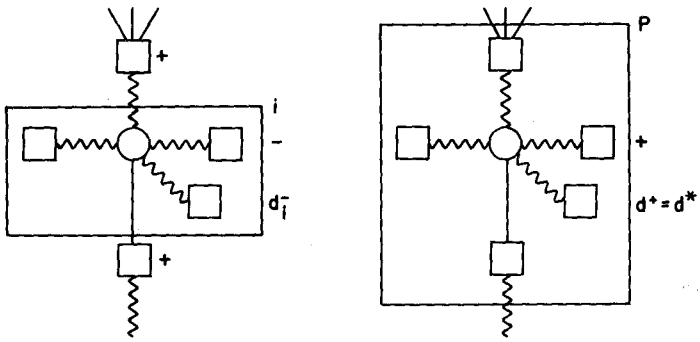

Fig. 4.7.

new pseudonode p should be an outer node with $d_p^+ = d^*$. Scan pseudonode p and return to the beginning of Step 1.

(c) The third type of blossom B has one original node and one non-matching edge as shown in Fig. 4.8. In this case, we form a new pseudonode of the outer node down toward the root together with the edge to the original node and incident matching edges. The other node j of B becomes an inner node with $d_j^- = d^*$. The remainder of this step is exactly the same as (A)(2).

Step 2. This step is reached with $d^* = \frac{1}{2}(d_i^+ + d_i^-)$ for some outer pseudonode i with edge k_i meeting node j in a different planted tree than i.

(A) There is an *augmenting path* including edge k_i and the planted forest; that is, a path with edges alternately matching and non-matching

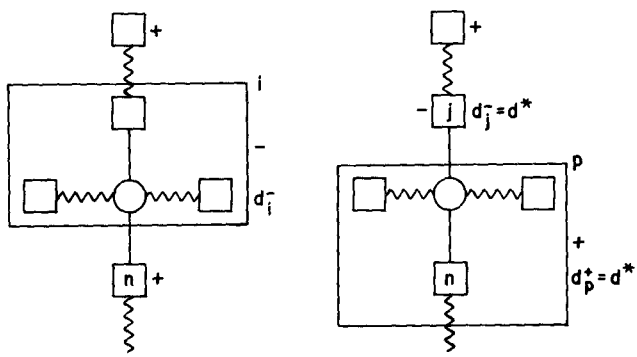

Fig. 4.8.

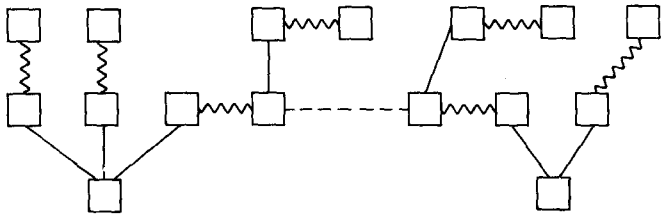

Fig. 4.9.

such that the ends of the path are deficient pseudonodes. The reason is that in a planted tree there is always an alternating path from an outer node to the root. Fig. 4.9 illustrates this situation. We augment by swapping matching and non-matching edges along the path as indicated in Fig. 4.9 by the line below the path. After augmenting, the deficient pseudonodes are no longer deficient. Remove both trees from the planted forest.

(B) For every node n of G_s, if both d_n^+ and d_n^- are greater than or equal to d^*, leave y_n as it is. Otherwise, n is in a planted tree. If n is an outer node, then $d_n^+ < d^*$. Replace y_n by $y_n + (d^* - d_n^+)$, and subtract $d^* - d_n^+$ from c'_e for every edge e (matching or non-matching) which meets node n. If n is an inner node, then $d_n^- < d^*$. Replace y_n by $y_n - (d^* - d_n^-)$ and add $d^* - d_n^-$ to c'_e for every edge e (matching or non-matching) which meets node n.

If there are no more deficient pseudonodes, go to Step 3. Otherwise, for each node n let

$$d_n^- = \min_e \{d_i^+ + c'_e : e \text{ meets an outer node } i\ (\neq n) \text{ and } e \text{ meets } n\}$$

and let k_n be the e giving the above minimum. If no such e exists, let $d_n^- = +\infty$ and $k_n = 0$. Return to Step 1.

Step 3. We now recover an optimum solution by specifying which x_e should be set equal to one. For edges e in G_s, let $x_e = 1$ if e is a matching edge. The pseudonodes of G_s will next be expanded to recover the blossom and to determine which e of the blossom should have $x_e = 1$. Then the pseudonodes of the blossoms will be expanded successively until all pseudonodes have been expanded. There are three types of blossoms, and we consider each type.

(A) A blossom may consist of an odd circuit as in Fig. 4.10. When

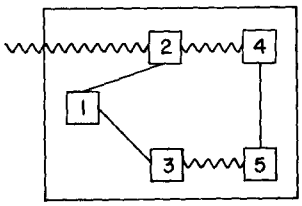

Fig. 4.10.

formed, node 1 in Fig. 4.10 was either deficient or incident to a matching edge meeting only one node of the blossom. However, any node may now be incident to such an edge, but only one node will be. From any node, there is an alternating path, beginning with a matching edge, to node 1. Thus, we can change the matching edges so that every node meets exactly one matching edge. For example, in Fig. 4.11, we can change the edges by swapping the matching and non-matching edges on the path 2, 4, 5, 3, 1 as illustrated in Fig. 4.11. Now let $x_e = 1$ for the matching edges of the blossom. All of the pseudonodes of the blossom can now be expanded.

(B) The second form of a blossom is a single original node together with any number of matching edges meeting it as illustrated in Fig. 4.12(a). When formed, the original node was either an odd node inci-

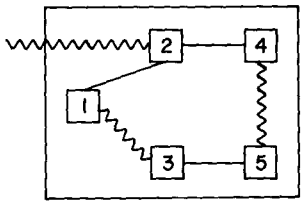

Fig. 4.11.

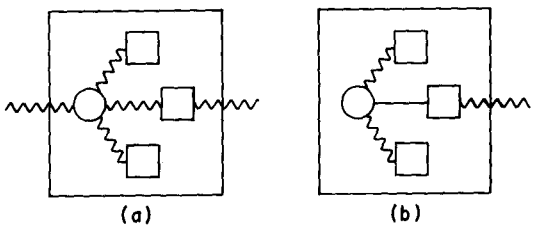

Fig. 4.12.

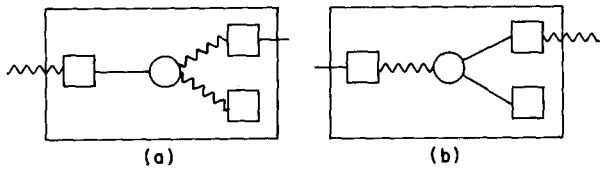

Fig. 4.13.

dent to no matching edge or was incident to a matching edge which met only that node of the blossom. If the matching edge meeting the pseudonode should still meet the original node, then let the edges of the blossom remain matching edges. However, if the incident matching edge meets a pseudonode as in Fig. 4.12(a), then change the edge of the blossom meeting that pseudonode to be non-matching as in Fig. 4.12(b). The pseudonodes of the blossom can now be expanded.

(C) The third form of a blossom is shown in Fig. 4.13(a). The original node meets one non-matching edge of the blossom and any number of matching edges. The pseudonode not incident to a matching edge of the blossom was either deficient or incident to a matching edge meeting only one node of the blossom when the blossom was formed. However, any node may now be incident to the matching edge which meets the blossom. If the original node is incident to it, then change the non-matching edge of the blossom to be a matching edge. If another pseudonode is met by it, as in Fig. 4.13(a), then swap the matching and non-matching edges as shown in Fig. 4.13(b).

The description of the algorithm is now completed.

There are several features of the algorithm which require some discussion before proving the optimality conditions (3.13) and (3.14).

First, let us show that the solution x_e, $e \in E$, does in fact satisfy (3.3), that is, odd nodes are met by an odd number of matching edges and even nodes by an even number. The augmentations in Step 2 are not directly concerned with original nodes of G since all of the nodes of the augmenting path are pseudonodes. Only when the pseudonodes formed by shrinking blossoms in Step 1(A)(1b) and (A)(2) are expanded do the original nodes come directly into view. An augmentation made in Step 3(B) of (C) keeps the same number (modulo 2) matching edges incident to the original node, except in case (B) when the pseudonode

was a deficient pseudonode. In that case, the pseudonode consists of a single odd node. When that pseudonode is expanded, the odd node will meet one matching edge. If the pseudonode is expanded in Step 1(C), then the odd node will henceforth meet an odd number of matching edges.

Clearly, the algorithm maintains the conditions that every pseudonode is incident to at most one matching edge and every matching edge meets at least one pseudonode. At the conclusion of the algorithm, there will be no more deficient pseudonodes, and every pseudonode will be incident to exactly one matching edge.

Next, we show that pseudonode variables y_n correspond to odd sets. In order to make this correspondence clear, define the *complete expansion* of a pseudonode p to be the subgraph of G obtained by successively expanding pseudonode p and all pseudonodes in that expansion until only original nodes remain. The complete expansion, then, has nodes S, which are a subset of the original nodes of G, and all edges meeting two nodes of S. We wish to show that every such S will be an odd set. Recall that an odd set is a set of nodes of G containing any number of even nodes and an odd number of odd nodes. This result could be proven by induction at the time pseudonode p is formed. Alternatively, the proof which follows uses the integrality of x_e and reveals again the motivation for the inequalities (3.5).

We have shown that, at the conclusion of the algorithm, odd nodes are met by an odd number of matching edges and even nodes are met by an even number of matching edges. In other words, condition (3.6) is satisfied: $\Sigma a_{ne} x_e \equiv b_n \pmod 2$. Adding for $n \in S$ gives

$$\sum \{x_e\colon \quad e \text{ meets } S\} \equiv \begin{cases} 1 \pmod 2 & \text{for } S \text{ an odd set}, \\ 0 \pmod 2 & \text{otherwise}, \end{cases}$$

by x_e being integer. In our case, the sum above is equal to one, and hence S must be an odd set.

Therefore, x_e, $e \in E$, satisfy (3.1), (3.2) and (3.6) and y_p do correspond to variables y_s of Section 3, where S is the node set of the complete expansion of p and is an odd set. We also know that x_e, $e \in E$, satisfy all of (3.5) because they satisfy (3.1), (3.2) and (3.6), and that (3.12) is true because every pseudonode is incident to one matching edge. It remains to be shown that y_p are non-negative, and that (3.8) and (3.11) hold.

The only time any y_n is lowered is in Step 1(B) or Step 2(B). In both

cases, no y_n could become negative because d^* is always limited by $y_n + d_n^-$ (see Step 1(γ)).

We now turn to the proof of (3.8):

$$\sum\{y_s\colon S \text{ incident to } e\} \leq c_e, \quad e \in E.$$

We use y_p and y_s interchangeably, where S is the node set of the complete expansion of p. Initially, the y_s are all zero and $c_e \geq 0$, so it is certainly true then. Also, it is clear that

$$c'_e = c_e - \sum\{y_s\colon S \text{ incident to } e\}.$$

since c'_e is changed by $-\Delta$ for all e meeting S whenever y_s is changed by Δ. Hence (3.8) is equivalent to $c'_e \geq 0$.

The algorithm may allow some c'_e to become negative in Step 1, but the changes in Step 2(B) will restore non-negativity. What will be shown is slightly more general: at any time that Step 1 is executed, changing y_n and c'_e as in Step 2(B) would restore $c'_e \geq 0$.

Suppose at the beginning of Step 1, using a previously assigned value of d^*, making the changes as in Step 2(B) would keep $c'_e \geq 0$. That change is to decrease c'_e by $d^* - d_n^+$ for outer nodes n and e meeting n and increase c'_e by $d^* - d_n^-$ for inner nodes n and e meeting n. The only e effected are in the surface graph G_s. Let us define c''_e to be this changed value. If edge e of G_s meets node i and j of G_s, then

$$c''_e = \begin{cases} c'_e + d_i^+ + d_j^+ - 2d^*, & i \text{ and } j \text{ outer nodes}, \\ c'_e + d_i^+ - d^*, & i \text{ outer and } j \text{ not planted}, \\ c'_e + d_i^+ - d_j^-, & i \text{ outer and } j \text{ inner}, \\ c'_e - d_j^- + d^*, & i \text{ not planted and } j \text{ inner}, \\ c'_e, & i \text{ and } j \text{ not planted}. \end{cases}$$

The only c''_e which decrease as d^* increases are the first two cases: i and j outer, i outer and j not planted.

If i and j are outer nodes, then when i becomes an outer node, it was scanned so $d_j^- \leq d_i^+ + c'_e$. Hence,

$$c''_e = c'_e + d_i^+ + d_j^+ - 2d^* \geq d_j^- + d_j^+ - 2d^*,$$

and $c''_e \geq 0$ provided

$$d^* \leq \tfrac{1}{2}(d_j^+ + d_j^-) .$$

Case (β) of Step 1 assures that d^* will satisfy this inequality.

If i is an outer node and j is not in a planted tree, then when i became an outer node, it was scanned, so as before $d_j^- \leq d_i^+ + c_e'$. Hence,

$$c_e'' = c_e' + d_i^+ - d^* \geq d_j^- - d^* ,$$

and $c_e'' \geq 0$ provided $d^* \leq d_j^-$. The increase in d^* in Step 1 is limited by case (α), which assured the above inequality.

In Step 1, some y_n are changed in (A)(2) and (B) when a blossom is shrunk. In addition to the above argument, we must show that those changes do not make any c_e'' negative. The change in Step 1(A)(2) is easily justified. There, y_n is increased by $d^* - d_n^+$. But that change is precisely the same as made in forming c_e'', so c_e'' does not change at all. Furthermore, edges meeting two nodes of the blossom will have $c_e' = c_e''$ after making this change. An edge e meeting only one node of the blossom will meet the new pseudonode p and will have c_e'' unchanged since $d_p^+ = d^*$ so $d^* - d_p^+ = 0$. The pseudonode p will be scanned at this point. Henceforth, $d^* - d_p^+$ enters into the expression for c_e'' for any edge e meeting pseudonode p. In Step 1(B), the changes are again exactly the changes in c_e' made in forming c_e''. The new pseudonode p has $d_p^+ = d^*$, as in Step 1(A)(2), so $d^* - d_p^+ = 0$.

Condition (3.11) remains to be shown; that is, $c_e' = 0$ for any matching edge. This condition will be shown by proving that $c_e'' = 0$ for any edge e in a planted tree, and $c_e' = 0$ for any edge e in a blossom. The only time an edge is made a matching edge is in an augmentation in Step 2 or in an expanding a pseudonode.

In Step 1(A)(1), when the planted forest is extended to include edge $k = k_i$, $c_k'' = 0$ since $d_n^+ + c_k' = d_i^-$, where k meets node n, an outer node, because edge k_i was so designated when d_i^- was determined. In both cases, (A)(1a) and (A)(1b), the matching edge adjoined to the planted forest has $c_e'' = 0$ as well.

The only other time the planted forest becomes larger is in Step 1(C) when a pseudonode is expanded and some or all of the blossom adjoined to a planted tree. The edges of the blossom all have $c_e' = 0$, and $c_e'' = 0$ as assured by setting $d_i^+ = d_j^-$ for the two incident nodes i and j (see Figs. 4.6, 4.7 and 4.8).

Consider now the case of a blossom forming and being shrunk. We wish to show that $c_e' = 0$ for every edge e of the blossom. With one ex-

ception, this fact follows from making the changes in y_n so that $c'_e = c''_e$. The one exception is the edge $k = k_i$ in Step 1(B) because that edge is not in the planted forest. The value of c'_k, after the changes in y_i and y_j, is

$$c'_k - (d^* - d_j^+) - (d^* - d_i^+) = d_i^- + d_i^+ - 2d^*$$

because $d_i^- = c'_k + d_j^+$. But $d^* = \frac{1}{2}(d_i^- + d_i^+)$, so c'_k becomes zero after the change.

5. Euler tours

In Section 3, we saw that matching theory provides us with an algorithm for the minimum length way to adjoin duplicate edges of G to form a graph G' having even degree. In Section 4, an algorithm to do so was given in detail. However, given an even-degree graph G', we are still left with the problem of actually finding an Euler tour, which is known to exist in G'. This section will be concerned with graphs G in which there exist Euler tours and will give methods for finding such a tour.

A tour was defined to be an alternating sequence

$$(n_1, e_1, n_2, e_2, n_3, \ldots, n_l, e_l, n_{l+1} = n_1)$$

of nodes and edges such that edge e_i meets the (distinct) nodes n_i and n_{i+1}. Before discussing algorithms, two ways of representing tours will be described. The two main algorithms produce these two different representations of an Euler tour. The edge-pairing representation is similar to the alternating sequence definition just given. That sequence can be thought of as providing the next edge e_{i+1} by which to leave node n_{i+1} given that we reached node n_{i+1} by edge e_i. That is, the edges e_i and e_{i+1} meeting node n_{i+1} are thus paired together. The first description of a tour is to provide for each node a list $(e_1, e_2), \ldots, (e_i, e_{i+1})$ of all such ordered pairs of edges meeting at that node.

The next-node representation is to list for each node n, the nodes, in order, that we go to when leaving node n. Let this list be denoted $L_n(1), L_n(2), \ldots, L_n(k)$. The first time node n is reached, leave it by going to node $L_n(k)$. The second time node n is reached, leave it by going to node $L_n(k-1)$.

Both representations require, in addition, the starting node. The edge-

pairing representation must treat the first and last edges of the Euler tour in a special way. Our convention will be that the starting node will include in its list of edge pairs the pairs $(0, e_1)$ and (e_k, ∞), where e_1 is the first edge of the tour and e_k is the last edge of the tour.

A tour can be easily followed given either description. However, to convert one description to the other requires actually tracing out the tour. Furthermore, the only way to tell whether the edge-pairing representation actually gives a tour is to try to follow the tour specified. Theorem 5.1 gives a necessary and sufficient condition for the next-node representation to give an Euler tour.

The first algorithm gives the end-pairing representation of an Euler tour, and the second algorithm gives the next-node representation. Although the algorithms are similar, they give different descriptions of possibly different Euler tours. The first algorithm obviously gives an Euler tour, but the proof that the second one does so is not trivial.

Both algorithms begin by tracing out a simple tour which may not include all edges. Then, begin at any node n_0 of the tour incident to edges not in the tour and complete a second simple tour not including any edge of the first one. The two algorithms move through the graph in the same way but produce different tours represented differently. In the first algorithm, if e_0 is the first edge leaving node n_0 in the second tour and e_L is the last edge entering node n_0, then for any edge pair (e_1, e_2) of the first tour meeting node n_0, swap the edge-pairings by replacing (e_1, e_2) by (e_1, e_0) and (e_L, e_2). This swap has the effect of interjecting the second tour into the first one. The new tour formed will follow the first one until edge e_1 is traversed to reach node n_0, then follow the second one until it is completed, and then follow the first one again. The new tour is longer than the first one. This procedure can be repeated until every edge is in the tour.

For completeness and for comparison with the second algorithm, we restate this algorithm in more formal terms.

5.1. *End-pairing algorithm*

Step 0. Let r be any node. Let both n_0 and n initially be equal to r. Let e_r be any edge meeting node r, and let e_1 be equal to 0, e_2 be equal to ∞, and e_0 be equal to e_r. All edges are unpaired. Let e be equal to e_r. Go to Step 1.

Step 1. Let n' be the node other than n incident to edge e. If there is an edge e' with an end meeting n' which is not yet paired, go to Step 2.

Otherwise, n' must be equal to n_0. In that case, form the edge pairs (e_1, e_0) and (e, e_2) meeting node n_0. Go to Step 3.

Step 2. Pair the edges e and e' meeting n'. Change n to be n' and e to be e'. Go to Step 1.

Step 3. Change n_0 to be any node which has at least one pair (e_1, e_2) of edges meeting it and at least one unpaired edge also meeting it. Let e_0 be an unpaired edge meeting n_0. Let n be set equal to n_0 and e be set equal to e_0, and go to Step 1. If no such node n_0 exists, terminate.

The second algorithm will produce a description of an Euler tour by giving a list of nodes $L_n(1), L_n(2), \ldots, L_n(k)$ to go to when leaving node n. In the way it moves through the graph, it is exactly the same as the preceding algorithm. However, the Euler tour produced may be different. For one thing, the edges will be traversed in the opposite direction from the way they are traversed during the algorithm.

5.2. *Next-node algorithm*

Step 0. Let r be any node. Let n_0 and n initially be equal to r. Let $k_i = 0$ for all nodes i. Let e be any edge meeting node n. All edges are unused.

Step 1. Let m be the node other than n incident to edge e. Let k_m be changed to k_{m+1} and for this new value of k_m let $L_m(k_m) = n$. Edge e is now used. If node m has any incident, unused edges, go to Step 2. Otherwise, node m must be equal to n_0. In that case, go to Step 3.

Step 2. Let n be changed to m and let e be any unused edge incident to m. Go to Step 1.

Step 3. Let n_0 be any node which has at least one used edge meeting it but with at least one unused edge meeting it, say edge e. Let n be changed to n_0 and go to Step 1. If no such node n_0 exists, terminate.

To follow an Euler tour using the next-node representation produced by the above algorithm, the first time node n is reached we will leave it by going to node $L_n(k_n)$ which is the last node we reached node n from in Step 1 of the algorithm. The second time n is reached, we will leave it by the next to last node we reached it from in Step 1. Notice that the direction specified for an edge to be traversed by the Euler tour is from n to $L_n(i)$ and is opposite from the direction the edge was traversed during the algorithm.

The proof that this algorithm will produce a next-node representation

of an Euler tour will be in two parts summarized by the two theorems below.

Theorem 5.1 [1]. *Let r be any node of an even, connected graph G. Next-node lists for G describe an Euler tour if and only if all of the following hold:*

(i) *the list* $L_n(1), \ldots, L_n(k_n)$ *for every node n has length* k_n *equal to one-half of the degree of n;*

(ii) *the number of edges meeting both nodes n and m is equal to the number of times* $n = L_m(i)$ *plus the number of times* $m = L_n(i)$;

(iii) *the edges* $(n, L_n(k_n))$, *for* $n \neq r$, *form an arborescence with root r, where the direction of each edge is from n to* $l_n(k_n)$.

An *arborescence* T with root r is defined to be a tree with a direction assigned to each edge so that every node n except $n = r$ has exactly one edge of T directed away from n, and node r has no edges of T directed away from it.

Theorem 5.2. *The next-node lists created by the next-node algorithm satisfy Theorem* 5.1 (i), (ii) *and* (iii).

Proof of Theorem 5.2. Condition (i) is clear because every time we reach node m in the algorithm, we use two edges and add one more entry $L_m(k_m)$ to the next-node list. Condition (ii) is true because once an edge is used it appears in a next node list.

Condition (iii) is easy to verify from the fact that the edge $(n, L_n(1))$, $n \neq r$, is the edge which node n is first reached by in the algorithm. Whenever a new addition is made to the next-node list for node n, the new node becomes $L_n(k_n)$, $k_n \geq 2$. The entry $L_n(1)$ never changes.

Thus, the edges $(n, L_n(1))$ remain fixed, and every time a node is first reached during the algorithm a new edge is adjoined to the collection T of $(n, L_n(1))$. At the time edge $(n, L_n(1))$ is first adjoined to T when node $n \neq r$ is first reached, it is the only edge of T meeting node n and becomes the unique edge of T directed away from n. That T is spanning is assured by the fact that Step 3 always begins with a node n_0 which has been previously reached so that it is in the arborescence.

Proof of Theorem 5.1 (see [1] and [3, p. 169]). First, conditions (i), (ii) and (iii) are true for an Euler tour. (i) and (ii) are obviously true since an Euler tour leaves a node once each time it enters it and uses

each edge once. It is also obvious that the collection T of edges $(n, L_n(k_n))$ from n to $L_n(k_n)$ for $n \neq r$ includes exactly one edge directed away from n for $n \neq r$. Every set S of nodes not containing r must have at least one edge of T directed from a node of S to a node not in S. Otherwise, the Euler tour would eventually reach S and stay in S. Therefore, T must form a connected graph including every node of G. Since T has one fewer edge than nodes of G and is connected, it must be a tree. Hence, (iii) is true.

Consider the converse. Suppose (i), (ii) and (iii) are true. If we begin at node r and follow the tour specified by the next-node lists, by (i) and (ii) we will eventually return to r having traversed every edge meeting r. Suppose this tour fails to be an Euler tour because some edges have not been traversed. Consider the set S of nodes incident to edges which have not been traversed. In particular, for $n \in S$ the edge $(n, L_n(1))$ will not have been traversed since that is the last edge specified by the next-node list. Also, $r \notin S$ by the previous remarks.

By (iii), the edges $(n, L_n(1))$, $n = r$, form an arborescence T. For any $n \in S$ there is a unique path from n to r in T. Along this path there is a node of S closest to r; that is, a node $n \in S$ such that $m \notin S$ for m on the path from n to r in T. In particular, $m = L_n(1) \notin S$, so every edge meeting m has been traversed. Therefore, edge $(n, L_n(1))$ has been traversed. Thus, a contradiction is reached since for $n \in S$, $(n, L_n(1))$ has not been traversed. Therefore, S must be empty and the tour must be an Euler tour.

The advantage of the next-node algorithm is that it offers opportunities to improve the algorithm not afforded by the edge-pairing algorithm. In the next section, we present an improvement of the algorithm and some variations of the Euler tour problem. The improvement on the algorithm is based on the fact that the algorithm can be terminated with a partial next-node representation. While considering this improvement, a more general problem will be treated. For now let us discuss some other algorithms for the Euler tour problem.

A method sometimes given for finding an Euler tour is an "isthmus-avoiding" algorithm [3]. This algorithm is to begin at any node r and take any edge not yet used as long as removing the edge from the set of unused edges does not disconnect the graph consisting of the unused edges and incident nodes to them. That this algorithm will produce an Euler tour is not obvious and is an interesting theorem. Practically, however, the algorithm would require an excessive amount of work be-

cause of the necessity of repeatedly determining whether or not an edge is an isthmus in the graph of unused edges.

Our first two algorithms specified an Euler tour without actually traversing it. In order to actually traverse an Euler tour, there is an interesting variation of the next-node algorithm which could be called a "maze search" Euler tour algorithm. This algorithm traverses each edge twice. The edges, in the order which they are traversed the second time, form an Euler tour.

If the postman was told which streets are on his route without being given a map, he could use this algorithm to find an Euler tour while walking over every street exactly twice.

5.3. *Maze-search algorithm*

Step 0. Let r be any node. Let n initially be equal to r. Let $k_i = 0$ for all nodes i. Let e be any edge meeting node n. Let $t_i = 0$ for all nodes i.

Step 1. Let m be the node other than n incident to e. Let k_m be changed to k_{m+1} and let $L_m(k_m) = n$ for this new value of k_m. Edge e is now *used*. If node m has any incident, unused edges, go to Step 2. Otherwise, go to Step 3.

Step 2. Let n be changed to be equal to m and let e be any unused edge meeting node m. Go to Step 1.

Step 3. Let $l = L_m(k_m - t_m)$ and then change t_m to be t_{m+1}. If there are any unused edges meeting node l, let e be such an edge, let n be equal to l, and go to Step 1. Otherwise, let m be equal to L and repeat Step 3 $m = r$ and $k_m = t_m$ in which case, terminate.

The postman traverses each edge twice. The first time he does so, he indicates at the node reached that that edge is to be traversed the next time in the opposite direction and records this edge as being the last edge by which the node is reached. He leaves the node by any unused edges if there are any. Otherwise, he leaves it by the last edge with which he reached the node and which has not been traversed twice. Every time he reaches a node, he must check for unused edges.

This algorithm is actually a special implementation of the next-node algorithm because Step 3 here is a way to carry out Step 3 of that algorithm while simultaneously traversing the final Euler tour. This algorithm would be a way to carry out the next-node algorithm if one must move from node to node along an edge and cannot obtain node information until the node is reached.

6. The mixed Euler tour problem

An edge is *directed away from* node i *toward* node j if it meets nodes i and j and must be traversed from i to j in any tour. Previously, all edges have been *undirected*: they could be traversed in either direction. A *mixed* graph is a graph in which some edges are directed and some are undirected. A postman may encounter a mixed graph if he has some one-way streets on his route.

In case all edges are undirected, the algorithm given in this section is an improvement on the next-node algorithm of Section 5. The algorithm here can be used for mixed graphs which are connected, even degree, and symmetric.

A *symmetric* mixed graph is a mixed graph such that every node n has the same number of edges directed away from n as directed toward n. Fig. 6.1 shows a symmetric graph.

The degree of a node of a mixed graph is the total number of edges, regardless of direction, meeting the node. An *even mixed graph* is a mixed graph with each node having even degree. A mixed graph is connected if it is connected when the directions on the edges are ignored.

When all of the edges of a connected graph are directed and the graph is symmetric, there is a particularly simple and attractive algorithm for specifying an Euler tour. The algorithm begins by finding a spanning arborescence of G. Recall that an arborescence with root node r is a tree T such that every node n of T, except $n = r$, has precisely one edge of T directed away from n.

Finding a maximal arborescence T of a directed graph G is easy. Given an arborescence T, adjoin to T any edge directed toward a node of T and away from a node not in T. When no such edges exist, the arborescence is maximal. Further, any maximal arborescence T will be spanning if G is symmetric, connected and directed. The reason is that

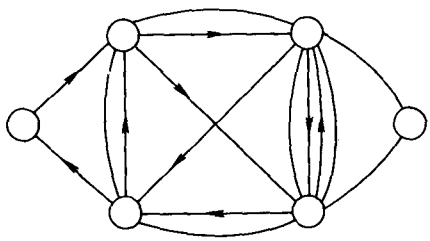

Fig. 6.1.

every set S of nodes will have the same number of edges directed away from a node of S and toward a node not in S as edges directed toward a node of S and away from a node not in S. Thus, every arborescence can be extended to a spanning arborescence in an easy way.

The algorithm to find an Euler tour in a directed, symmetric, connected graph G is to first find a spanning arborescence of G. Then, at any node n, except the root r of the arborescence, specify any order for the edges directed away from n so long as the edge of the arborescence is last in the ordering. For the root r, specify any order at all for the edges directed away from r.

This algorithm was used by van Aardenne-Ehrenfest and de Bruin to enumerate all Euler tours in a certain directed graph [1]. Before showing that it will produce an Euler tour, we introduce a slightly more general case.

The first mixed graphs which we will consider are graphs with both directed and undirected edges such that the directed edges form a symmetric, connected spanning subgraph of G. Every node is, then, met by two or more directed edges, and the directed edges form a connected graph. The undirected edges are required to be an even degree subgraph of G but need not be connected.

In such a graph, a maximal arborescence is still spanning. Given a spanning arborescence, the undirected edges meeting a given node can be ordered in any way and the directed edges away from a given node n can be ordered in any way as long as the edge of the arborescence directed away from n is last in the ordering.

With this ordering of undirected and directed edges at each node n, an Euler tour is found by beginning at the root r of the arborescence and following the rule below.

Rule. Whenever node n is reached, leave node n by the next unused, undirected edge if one is available. If all undirected edges have been used, then leave node n by the next unused edge directed away from n.

We now prove that this rule will produce an Euler tour. The Euler tour begins at root r and leaves by an undirected edge if any meet r. The node reached must have an unused, undirected edge because there are an even number of undirected edges meeting each node. Hence, we will use an undirected edge to leave that node. This process will continue, using only undirected edges, until node r is returned to and there are no remaining unused, undirected edges meeting r. At that point, we leave r

by using an edge directed away from r (there is at least one). The node n reached may have unused, undirected edges which meet it. If so, we will use undirected edges until eventually returning to n with no more unused, undirected edges meeting n. At that point, leave n using an edge directed away from n but not in the arborescence unless that edge of the arborescence is the only edge directed away from n. Each time a node n is first reached by an edge directed toward n, we will use all of the undirected edges meeting n before continuing with a directed edge, directed away from n.

To show that every edge is traversed is easy using Theorem 5.1. If we follow a tour specified by the rule and record the next-node description of the tour, the three conditions of the theorem are satisfied. The only difficulty is showing condition (iii), and it is satisfied because we start with a spanning arborescence from the directed edges.

Consider now a more general case. We drop the requirement that the directed edges form a spanning, connected subgraph of G and only require that G itself be connected, even, and, as before, that G be symmetric. The algorithm for finding an Euler tour in G will consist of assigning directions to enough undirected edges so that the resulting mixed graph is still balanced and so that the directed edges are spanning and connected.

To avoid confusion, an undirected edge is called *assigned* if we have specified a direction from the edge. A node will be called *virgin* if we have not yet encountered it in the algorithm. Initially, all nodes are virgin.

A substep of the algorithm is to *extend the arborescence* using directed edges, which means to look at all edges directed toward a specified node n and see if the other end of such an edge meets a node m which is virgin. If so, include edge (m, n) in the arborescence and extend the arborescence from node m. Node m then becomes a *reached* node and loses its virgin status.

Another substep is to *scan* a node n, which means to look at all undirected edges e meeting node n and see if the other node m incident to e is virgin. If so, assign e the direction away from m toward n, adjoin edge e to the arborescence, and move to node m (let n be set equal to m).

The algorithm is given below.

Step 0. Label all nodes as virgin nodes. Choose any node r, let n be set equal to r, and go to Step 1.

Step 1. If n is virgin, relabel it as *reached* instead of virgin and, starting at node n, extend the arborescence using directed edges. If $n = r$ and the arborescence becomes spanning, terminate. Otherwise, go to Step 2.

Step 2. If n is reached, scan it. If a virgin node m is found, go to Step 1 (with n set equal to m). Otherwise, n becomes *scanned*, and go to Step 3.

Step 3. When n is scanned, let e be any unassigned, undirected edge meeting n. Let m be the other node incident to e. Assign e the direction away from m and toward n, let n be set equal to m, and go to Step 2 (m cannot be virgin if n is scanned) unless m is equal to r and the arborescence is spanning in which case terminate.

Every node is either virgin, reached, or scanned. The algorithm terminates when there are no more virgin nodes and when $n = r$. Clearly, no remaining virgin nodes is equivalent to the arborescence being spanning.

At termination, every node is still symmetric, considering both directed and assigned edges, because we return to r and along the way assign an edge away from each node n and then toward n. Furthermore, the directed and assigned edges form a spanning subgraph of G because the arborescence is spanning. The undirected, unassigned edges still have even degree at each node because an even number of undirected edges meeting a given node have been assigned a direction. Thus, the rule previously given will now enable one to actually trace out an Euler tour.

If a graph G has only undirected edges and is even degree and connected, then the algorithm just given can skip Step 1. As such it is a modification of the algorithm given in Section 5. This algorithm allows earlier termination and does not actually traverse an Euler tour. It simply assigns enough edges to make the assigned edges be spanning while maintaining symmetry. However, an Euler tour is not specified in the form of any of the three defined forms of Section 5. The rule given in this section does not allow us to say that an Euler tour can be easily traversed given the output of this algorithm.

A pointer can be used in Step 2 to keep track of where we are in the scanning process. Say $P(n)$ is the last undirected edge meeting node n which we have scanned. Initially, then, all $P(n) = 0$. Each time we look at an edge in going through the list of undirected edges, $P(n)$ should be increased by one. The same pointer can be used in extending the arborescence in Step 1. Steps 2 and 3 can be accomplished with an amount

of work growing linearly in the number of edges which are assigned directions.

A more general problem has been treated by Ford and Fulkerson [10, Section II.7]. They remove the restriction of symmetry and show that an Euler tour exists if and only if there is no subset S of nodes such that more edges are directed away from a node in S toward a node not in S than there are total other edges meeting a node in S and a node in S. Of course, connectedness and even degree are also required. An Euler tour can be found by first assigning directions to enough edges so that symmetry is achieved. This problem is a network flow problem (see [10, p. 60]) and the network flow methods either succeed or terminate with a set S as above. Once symmetry is achieved, the algorithm just given can be used.

7. The mixed postman problem

A postman tour in a mixed graph G is a tour of G using every edge at least once. Here, a tour must traverse the directed edges of G in the specified direction. The mixed postman problem is to find the minimum length postman tour, where the length of a tour is, as before, the sum of the lengths c_e of the edges e. As before, $c_e \geq 0$ is assumed.

When all edges of G are undirected, the problem is the Chinese postman problem, which we have already treated. When all edges of G are directed, the problem is easier than the undirected case and can be solved using network flows. The problem is solved by first symmetrizing every node, and then using the algorithm in Section 6 to find an Euler tour of the symmetric graph and a postman tour of the original graph. The symmetrizing problem is to duplicate edges one or more times so that every node becomes symmetric and minimize the sum of the lengths of the duplicated edges. Let x_e be the additional duplicate copies of edge e to be adjoined to the graph. The problem is to minimize $z = \Sigma c_e x_e$ subject to $x_e \geq 0$ and integer for $e \in E$, and

$$\sum \{x_e : e \text{ directed away from } n\} - \sum \{x_e : e \text{ directed toward } n\} = b_n, \quad n \in N,$$

where b_n is the number of edges toward n minus the number of edges away from n. The above is a standard min-cost flow problem.

By contrast with the undirected postman problem, the directed postman problem may not have a solution even when G is connected. In the undirected case, connectedness of G assures that there is a path from any node to any other node. In the directed case, we need a directed path, or chain, between every pair of nodes. Equivalently, using a network circulation theorem [11], the above flow problem has a solution if and only if there is no proper subset S of nodes such that every edge meeting one node i in S and one node j not in S is directed away from i and toward j. Network flow algorithms can be used to solve the directed postman problem or discover such a set S when there is no solution.

For the mixed postman problem, an analogous, necessary and sufficient condition for the existence of a postman tour can be given. There exists a postman tour if and only if there is no proper subset S of nodes such that every edge meeting one node in S and one node not in S is directed away from the node in S. This condition is obviously necessary. To prove sufficiency, the algorithm of Section 4 can be used to duplicate edges so as to make all degrees even. Once the degrees are even and this condition is satisfied, we will show next how to duplicate edges so as to symmetrize the graph while maintaining even degree. In fact, we can do this latter step in an optimum way once the degrees are all even. However, these two steps may not produce an optimum answer to the mixed postman problem because the optimum way to duplicate edges so as to make even degrees may not be the best way to do so if we later have to duplicate more edges to symmetrize the graph.

Consider now the mixed postman problem, where every node is met by an even number of edges (directed or undirected). In cities, where a postman may face the situation in which some edges are directed (one-way streets) and some are undirected, nodes are often of degree four since an intersection has four streets coming into it. Thus, the even degree case may be interesting although it is easier to solve than the undirected postman problem, where the degree of some nodes may be odd.

Let U denote the undirected edges and D the directed edges. For $e \in D$, x_e will denote the number of additional copies of e to adjoin to G. Form the set U_1 of directed edges by putting two edges in U_1 for each edge $e \in U$: one edge e^1 directed away from i and toward j and one edge e^2 directed toward i and away from j, where edge e meets nodes i and j. Form the set U_2 in the same way. The sets U_2 and U_1 are disjoint. For $e \in U$, there are two directed copies h and k of e in U_1 and two directed copies f and g in U_2. Let x_h be the additional, directed

copies of edge e to be adjoined to G and let x_k be the additional, oppositely directed copies of e to be adjoined to G. For $f \in U_2$, $x_f = 1$ will be interpreted to mean that e (the original e, not a copy) should be directed as f is and $x_g = 1$ means that e should be directed as g is. Either x_f or x_g or both can be zero. Our purpose here is not to direct all undirected edges but only to form a symmetric graph from G by directing and perhaps duplicating undirected edges of G and duplicating directed edges. The directed edges are considered as already present.

Let $E' = D \cup U_1 \cup U_2$. The network flow problem to be solved is to minimize z subject to

$$z = \sum_{e \in D} c_e x_e + \sum_{e \in U_1} c_e x_e , \tag{7.1}$$

$$x_e \text{ a non-negative integer for } e \in D \cup U_1 , \tag{7.2}$$

$$x_e = 0 \text{ or } 1 \text{ for } e \in U_2 , \tag{7.3}$$

$$\sum \{x_e : e \in E' \text{ is directed away from } n\} - \sum \{x_e : e \in E' \text{ is directed toward } n\} = b_n , \quad n \in N, \tag{7.4}$$

where b_n is the number of edges e in D directed toward n minus the number of edges e in D directed away from n.

A solution to (7.1)–(7.4) will produce the minimum cost symmetric graph G' formed by directing some undirected edges of G and perhaps duplicating edges. Both $x_h + x_f$ and $x_k + x_g$ will not be positive at the same time in an optimum solution because if they were, we could lower both, maintain (7.4) and reduce; or leave unchanged, z given by (7.1) because $c_e \geq 0$.

The even degree condition, which was assumed for G, has not been shown for G'. If G' is even, then we have solved the even, mixed postman problem.

Even degree in G' is equivalent to

$$\sum \{x_e : e \in D \cup U_1 \text{ and } e \text{ meets } n\} \equiv 0 \pmod 2 ,$$

that is, the sum of duplicated edges meeting node n is even. The number of original edges meeting node n is even by assumption, so even degree of the duplicated edges is needed. We will show that an optimum solution to (7.1)–(7.4) satisfies this additional requirement.

Form U_3 from U by taking one directed copy of each $e \in U$. In U_2 there are two directed copies f and g. Let $h \in U_3$ be a copy of $e \in U$ and directed as f is. Let

$$x_h = 1 + x_f - x_g.$$

Since x_f and x_g do not appear in (7.1) because f and g are in U_2, they only appear in (7.3) and (7.4), and in (7.4) they always appear as $x_f - x_g$ or $-(x_f - x_g)$. We can thus substitute $x_f - x_g = x_h - 1$ into (7.3) and (7.4) to obtain

$$x_h = 0, 1 \text{ or } 2, \quad h \in U_3, \tag{7.3'}$$

$$\sum\{x_e : e \in E'' \text{ is directed away from } n\} - \sum\{x_e : e \in E'' \text{ is directed toward } n\} = b'_n, \quad n \in N, \tag{7.4'}$$

where $E'' = D \cup U_1 \cup U_3$ and where b'_n is the number of edges in $D \cup U_3$ directed toward n minus the number of edges in $D \cup U_3$ directed away from n.

The problem (7.1), (7.2), (7.3'), (7.4') is equivalent to (7.1)–(7.4). The important fact about this formulation is that b'_n is even for all $n \in N$. The reason is that the number of edges in $D \cup U_3$ meeting node n is even and taking any sum of that number with some +1 and some −1 coefficients will still be even. Furthermore, the bounds on the variables are just $x_e \geq 0$ for $e \in D \cup U_1$ and $0 \leq x_e \leq 2$ for $e \in U_3$. Therefore, every x_e, $e \in E''$, will be even. The reason is the same reason that integer capacities on edges and integer flow requirements give an integer answer.

In the resulting graph, formed by directing some edges of U and duplicating some edges, there will be an Euler tour, and the method described earlier applies. The resulting graph has an even number of edges meeting every node and each node is symmetric.

Several easy generalizations of the undirected Chinese postman problem can be mentioned. An edge e may not be required in the tour but may be permitted. Such an edge would not be used in determining which nodes are even or odd but would appear in the problem (3.6), (3.7), (3.8). On the other hand, an edge may be required in the tour but may be permitted to be traversed only once. In that case, the edge is used in determining even or odd nodes but does not appear in the problem (3.6), (3.7), (3.8). In this connection, notice that the problem

(3.1)–(3.4) can be solved regardless of how the nodes came to be designated as even or odd provided there are an even number of odd nodes and the graph is connected (or each connected component contains an even number of odd nodes).

An important application area which involves a generalization of the postman problem involves forming some fixed number of tours each of which must meet some requirement. That is, there is not just one postman, but instead a central post office with many postmen. The problem is to assign routes to the postmen using the fewest possible postmen with no postman having too long a tour. Similar problems include garbage collection [2], street cleaning, milk delivery, school bus scheduling, etc. In these problems, the service required is naturally associated with the edges of a graph rather than the nodes. Node-oriented problems of this type are traveling salesman problems or generalizations. Such problems are very difficult, and even the problem of finding a Hamiltonian tour is difficult. The corresponding edge problem is to find an Euler tour. Considerable work using "edge-oriented" methods has appeared [13, 15, 16] for multi-postman problems of an "edge" type.

References

[1] van Aardenne-Ehrenfest and N.G. de Bruijn, "Circuits and trees in oriented graphs", *Simon Stevin* 28 (1951) 203–217.

[2] E.J. Beltrami and L.D. Bodin, "Networks and vehicle routing for municipal waste collection", Report No. UPS 72-18, State University of New York, Stony Brook, N.Y. (1972).

[3] C. Berge, *The theory of graphs and its applications* (Wiley, New York, 1962).

[4] J. Edmonds, "Paths, trees and flowers", *Canadian Journal of Mathematics* 17 (1965) 449–467.

[5] J. Edmonds, "Maximum matching and a polyhedron with 0, 1-vertices", *Journal of Research of the National Bureau of Standards Section B*, 1, 2 (1965) 125–130.

[6] J. Edmonds, "The Chinese postman problem", *Operations Research* 13 Suppl. 1 (1965) 373.

[7] J. Edmonds and E.L. Johnson, "Matching: a well-solved class of integer linear programs", in: *Combinatorial structures and their applications* (Gordon and Breach, New York, 1970) 89–92.

[8] J. Edmonds, E.L. Johnson and S. Lockhart, "Blossom I: a computer code for the matching problem", to appear.

[9] L. Euler, "Solutio problematis ad geometriam situs pertinentis", *Commentarii Academiae Petropolitanae* 8 (1736) 128–140.

[10] L.R. Ford Jr. and D.R. Fulkerson, *Flows in networks* (Princeton Univ. Press, Princeton, N.J., 1962).

[11] A.J. Hoffman, "Some recent applications of the theory of linear inequalities to extremal combinatorial analysis", in: *Proceedings of Symposia on Applied Mathematics* Vol. 10 (American Mathematical Society, Providence, R.I., 1960).

[12] T.C. Hu, "Revised matrix algorithms for shortest paths in a network", *SIAM Journal* 15 (1967) 207–218.
[13] T.M. Liebling, *Graphentheorie in Planungs- und Tourenproblemen*, Lecture Notes in Operations Research and Mathematical Systems 21 (Springer, Berlin, 1970).
[14] K. Mei-Ko, "Graphic programming using odd or even points", *Chinese Mathematics* 1 (1962) 273–277.
[15] C.S. Orloff, "Routing and scheduling a fleet of vehicles to/from central facilities – the school bus problem", Ph. D. Thesis, Cornell University, Ithaca, N.Y. (1972).
[16] R. Stricker, "Public sector vehicle routing: the Chinese postman problem", Master Thesis, Massachusetts Institute of Technology, Cambridge, Mass. (August 1970).