*OMEGA*, The Int. Jl of Mgmt Sci., Vol. 1, No. 6, 1973

# The Optimum Traversal of a Graph

NICOS CHRISTOFIDES

Imperial College, London

*(Received May, 1973)*

For a given graph (network) having costs $[c_{ij}]$ associated with its links, the present paper examines the problem of finding a cycle which traverses every link of the graph at least once, and which incurs the minimum cost of traversal. This problem (called the *graph traversal* problem, or the *Chinese postman* problem [9]) can be formulated in ways analogous to those used for the well-known travelling salesman problem, and using this apparent similarity, Bellman and Cooke [1] have produced a dynamic programming formulation. This method of solution of the graph traversal problem requires computational times which increase exponentially with the number of links in the graph. Approximately the same rate of increase of computational effort with problem size would result by any other method adapting a travelling salesman algorithm to the present problem.

This paper describes an efficient algorithm for the optimal solution of the graph traversal problem based on the matching method of Edmonds [5, 6]. The computational time requirements of this algorithm increase as a low order (2 or 3) power of the number of links in the graph. Computational results are given for graphs of up to 50 vertices and 125 links.

The paper then discusses a generalised version of the graph traversal problem, where not one but a number of cycles are required to traverse the graph. In this case each link has (in addition to its cost) a quantity $q_{ij}$ associated with it, and the sum of the quantities of the links in any one cycle must be less than a given amount representing the cycle capacity. A heuristic algorithm for the solution of this problem is given. The algorithm is based on the optimal algorithm for the single-cycle graph traversal problem and is shown to produce near-optimal results.

There is a large number of possible applications where graph traversal problems arise. These applications include: the spraying of roads with salt-grit to prevent ice formation, the inspection of electric power lines, gas, or oil pipelines for faults, the delivery of letter post, etc.

## INTRODUCTION

IN MOST operations involving delivery of goods to customers by road, the positions of the customers are defined as single points either by their co-ordinates or as points in a road network [1, 2, 3]. In many situations, however, the customers are so numerous that identifying them individually is not a practical proposition. Such operations include the delivery of milk or post to

houses, where what is required is that every street in the network representing the town, say, should be covered (i.e. traversed at least once), by the milkman, postman, etc. Yet in other cases in which exactly the same basic problem occurs, the "customers" may in fact be infinite in number. Such a problem arises when spraying salt-grit on roads in winter in order to prevent ice formation. In this case only the roads to be sprayed form the corresponding road network to be covered, and every bit of these roads can be considered as a "customer". Other examples of the last type include all problems of inspecting linearly distributed systems such as electric power lines, gas or oil pipelines, etc.

The basic underlying theoretical problem in all these cases can be stated as follows.

*Problem A*

Given a graph (network) $G$ of $n$ vertices $x_1,x_2 \ldots, x_n$ and $m$ links $(x_i,x_j)$ connecting some of these vertices, find a cycle which traverses each and every link of $G$ at least once and which minimises the total cost incurred; the link costs being given by the matrix $C = [c_{ij}]$.

The apparent similarity of this problem to the well-known travelling salesman problem is quite obvious since this last problem requires that minimum cycle which passes through every vertex (instead of through every link) of $G$ once and once only (instead of at least once). Prompted by this similarity, Bellman and Cooke [1] have developed a dynamic programming algorithm for the solution of problem A which is very closely related to an earlier formulation of the travelling salesman problem as a dynamic program. Their method exhibits (just as for the travelling salesman problem) exponentially increasing computer time and storage requirements (with respect to $m$). The result is that graphs with more than approximately 15 links cannot be optimally solved on present-day computers.

In the present paper we give a graph-theoretic algorithm based on the matching method in [6], for the optimal solution of problem A, which is capable of dealing with graphs containing hundreds of links. Computational results are given for randomly generated graphs of up to 50 vertices and 125 links and it is seen that computing times increase as a low order monomial (and not exponentially) with $m$.

Just as the pure travelling salesman problem rarely appears in practice but its variants very often do, so with the present problem. Very often in practice, the graph $G$ is not traversed by a single person but by a number of people starting from some service-centre (vertex). If now we associate a quantity $q_{ij}$ with every link $(x_i,x_j)$ of $G$, then the cycle traversed by every person is constrained to have a total quantity (being the sum of the $q_{ij}$'s of the links in the cycle) less than or equal to a given capacity $W$. If a link is traversed by two or more cycles then the $q_{ij}$ of this link can be allocated to either (but not both) of the cycles, since only one cycle need supply this link. The cost $c_{ij}$ is, however,

incurred by both cycles since the cost depends only on whether the link is traversed or not. Thus, in the example of salt-grit spraying of roads mentioned earlier, $q_{ij}$ may be the amount of salt-grit required to spray a road $(x_i,x_j)$ and $W$ would then be the vehicle capacity.

Problem A can now be generalised as follows.

*Problem B*

Given a graph $G$ with link costs $C = [c_{ij}]$ and link quantities $Q = [q_{ij}]$ find a number of cycles starting from vertex $x_1$, which traverse every link of $G$ at least once so that the sum of the $q_{ij}$ of every link supplied by a given cycle is $\leqslant W$ and the total travelling cost is minimised.

The similarity of problem B to the vehicle dispatching problem [2] is quite clear. In this paper we will give a heuristic algorithm (based on our exact algorithm for problem A), which produces approximate but good quality answers to problem B.

## ALGORITHM FOR PROBLEM A

One of the first theorems in graph theory (due to Euler) is concerned with whether a given graph $G$ possesses a cycle which traverses each link of $G$ once and once only. If such a cycle (called an eulerian cycle) exists, then this cycle has a cost equal to the sum of the costs of all the arcs of $G$ (which is the minimum possible cost for a cycle covering $G$) and therefore problem A becomes simply the problem of finding such a cycle. Euler's theorem states that such a cycle exists if, and only if, the degree of every vertex of $G$ (i.e. the number of links incident at every vertex) is even.

Consider now the case of an arbitrary graph $G = (X, U)$ where $X$ is its set of vertices and $U$ its set of links. Of the vertices of $X$ some vertices (say in the set $X_e$), will have even degrees and some (in the set $X_o = X - X_e$) will have odd degrees. Now the sum of the degrees $d_i$ of all the vertices $x_i \epsilon X$ is equal to twice the number of links in $U$ (since each link adds unity to the degrees of its two end vertices) and is therefore an even number $2m$. Hence

$$\sum_{x_i \epsilon X} d_i = \sum_{x_i \epsilon X_e} d_i + \sum_{x_i \epsilon X_o} d_i = 2m \qquad (1)$$

and since $\sum_{x_i \epsilon X_e} d_i$ is even $\sum_{x_i \epsilon X_o} d_i$

is also even, which means that since all $d_i$ in this last expression are odd the number $| X_o |$ of vertices of odd degree is even.

Let $M$ be a set of paths of $G$ ($\mu_{ij}$ say) between end vertices $x_i$ and $x_j$ ($x_i$, $x_j \epsilon X_o$) so that no two paths have any end vertex the same, i.e. the paths are between disjoint pairs of vertices of $X_o$ and constitute a pairwise vertex

matching. The number of paths $\mu_{ij}$ in $M$ is $\frac{1}{2} \mid X_o \mid$, and since $\mid X_o \mid$ was shown to be always even, this number is always integer as, of course, it should be. Suppose now that all the links forming a path $\mu_{ij}$ are added into $G$ as *artificial* links in parallel with the links of $G$ already there. (In the first instance this means that all links of $G$ forming $\mu_{ij}$ are now doubled.) This is done for every path $\mu_{ij} \epsilon M$ and the resulting graph is called $G_o(M)$. Since some links of $G$ may appear in more than one path $\mu_{ij}$, some links of $G_o(M)$ may (after all the paths $\mu_{ij}$ have been added in) be in triplicate, quadruplicate, etc.

We can now state the following theorem.

*Theorem* 1

For any cycle covering $G$, there is some choice of $M$ for which $G_o(M)$ possesses an eulerian cycle corresponding to the cycle of $G$. The correspondence is such that if a cycle traverses a link $(x_i,x_j)$ of $G$ $\gamma$ times, there are $\gamma$ links (one real and $(\gamma-1)$ artificial) between $x_i$ and $x_j$ in $G_o(M)$, each of which links is traversed exactly once by the eulerian cycle of $G_o(M)$; and conversely.

*Proof*

If a cycle covers $G$, then at least one link incident at every odd-degree vertex $x_i$ must (by Euler's theorem), be traversed twice. (A link traversed twice can be considered as two parallel links, one real and one artificial, both of which are traversed once.) Let this link be chosen as $(x_i,x_k)$. If the degree $d_k$ of $x_k$ in $G$ is odd then the addition of the artificial link earlier will make $d_k$ even and only this link need be traversed twice as far as $x_i$ and $x_k$ are concerned. If, however, $d_k$ is even, then the addition of the artificial link will now make $d_k$ odd and a second link leading from $x_k$ must also be traversed twice (i.e. another artificial link added). The argument continues from $x_k$ until a vertex of odd degree is reached as mentioned above. Thus, in order to satisfy the condition of traversibility at $x_i$ a whole path from $x_i$ to some other odd-degree vertex $x_r$ of $X_o$ must be traversed twice. This automatically satisfies the traversibility condition of vertex $x_r$. Similarly for all other vertices $x_i$ of $X_o$, which means that a whole set $M$ of paths of $G$, as defined earlier, must be traversed twice, and since this means that every link of $G_o(M)$ must be traversed once, the theorem follows.

The algorithm for the solution of problem A follows immediately from the above theorem, since all that is now necessary is to find that set of paths $M_{min}$ (matching the vertices of odd degree), which produces the least additional cost. The least cost cycle covering $G$ would then have a cost equal to the sum of the costs of the links of $G$ plus the sum of the costs of the links in the paths of $M_{min}$. This is the same as the sum of the costs of all the links—real and artificial—of the graph $G_o(M_{min.})$ A description of the algorithm now follows.

*Step* 1

Using a shortest path algorithm [4] on the graph $G$ with cost matrix $[c_{ij}]$, form the $|X_o|$ by $|X_o|$ matrix $D = [d_{ij}]$ where $d_{ij}$ is the cost of the least-cost path from a vertex $x_i \epsilon X_o$ to another vertex $x_j \epsilon X_0$.

*Step* 2

Use Edmonds' [5] minimum matching algorithm to find that pairwise matching $M_{min}$ of the vertices in $X_o$ according to the cost matrix $D$.

*Step* 3

If vertex $x_\alpha$ is matched to another vertex $x_\beta$ identify the least-cost path $\mu_{\alpha\beta}$ (from $x_\alpha$ to $x_\beta$) corresponding to the cost $d_{\alpha\beta}$ of step 1. Insert artificial links in $G$ corresponding to links in $\mu_{\alpha\beta}$ and repeat for all other paths in the matching $M_{min}$ to obtain the graph $G_o(M_{min})$.

*Step* 4

The sum of the costs from matrix $[c_{ij}]$ of all links in $G_o(M_{min})$—taking the cost of an artificial link to be the same as the cost of the real link in parallel with it—is the minimum cost of a cycle covering $G$. The number of times that this cycle traverses a link $(x_i,x_j)$ being the total number of links in parallel between $x_i$ and $x_j$ in $G_o(M_{min})$.

It should be noted that both the shortest path and matching algorithms, used as elementary steps in the above algorithm, are efficient procedures so that the present method can also be expected to be efficient. It should also be noted here that since at step 2 we are using a minimum matching, no two shortest paths $\mu_{ij}$ and $\mu_{pq}$ in such a matching (of say, $i$ to $j$ and $p$ to $q$) can now have any link in common; for if they have link $(a,b)$ in common as shown in Fig. 1, then a matching of $i$ to $q$ (using subpaths $i$ to $b$ and $b$ to $q$) and a

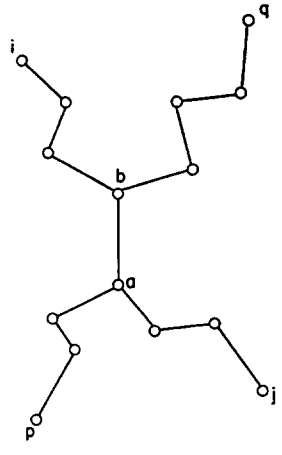



FIG. 1. *Paths $\mu_{ij}$ and $\mu_{pq}$ having common link $(a, b)$.*

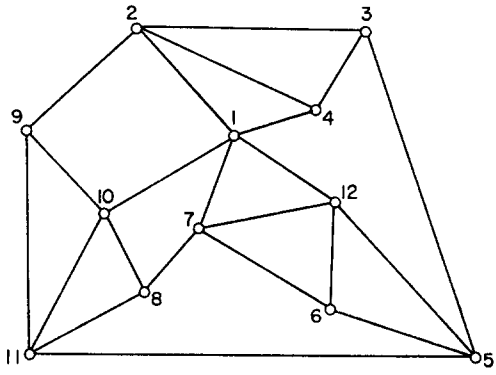



FIG. 2. *Graph for the examples.*

matching of $p$ to $j$ (using subpaths $p$ to $a$ and $a$ to $j$) produces an overall matching of cost $2c_{ab}$ less than the original matching, which is contrary to the assumption that this matching is minimal. This means that the graph $G_o(M_{min})$ does not have more than two links in parallel between any two vertices, i.e. the optimal cycle never traverses any link of $G$ more than twice.

## EXAMPLE A

Consider the graph $G$ of Fig. 2 having 12 vertices and 22 links where the link costs are as given in Table 1 (blank entries being taken as infinities). The problem is to find a cycle which covers $G$ and has least cost.

The set $X_o$ of odd degree vertices for this graph is {1, 3, 4, 6, 8, 9}.

TABLE 1. COST MATRIX FOR EXAMPLE A

$C =$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | X | 2 | | | | | | | | | | |
| 2 | 13 | X | 3 | | | | | | | | | |
| 3 | | 18 | X | 4 | | | | | | | | |
| 4 | 17 | 9 | 20 | X | 5 | | | | | | | |
| 5 | | | 5 | | X | 6 | | | | | | |
| 6 | | | | | 7 | X | 7 | | | | | |
| 7 | 19 | | | | | 4 | X | 8 | | | | |
| 8 | | | | | | | 8 | X | 9 | | | |
| 9 | | 2 | | | | | | | X | 10 | | |
| 10 | 19 | | | | | | | 3 | 16 | X | 11 | |
| 11 | | | | | 20 | | | 10 | 14 | 12 | X | 12 |
| 12 | 4 | | | | 11 | 3 | 18 | | | | | X |

Using Dijkstra's shortest path algorithm we find matrix $D$ of step 1 as

$D =$

| | 1 | 3 | 4 | 6 | 8 | 9 |
|---|---|---|---|---|---|---|
| 1 | X | 3 | | | | |
| 3 | 19 | X | 4 | | | |
| 4 | 17 | 20 | X | 6 | | |
| 6 | 7 | 12 | 24 | X | 8 | |
| 8 | 19 | 24 | 30 | 12 | X | 9 |
| 9 | 15 | 20 | 11 | 22 | 19 | X |

At step 2 the minimal matching algorithm matches the following vertices: (two other matchings are also minimal)

1 with 6, path: 1–12–6
3 with 8, path: 3–5–6–7–8
9 with 4, path: 9–2–4 . (2)

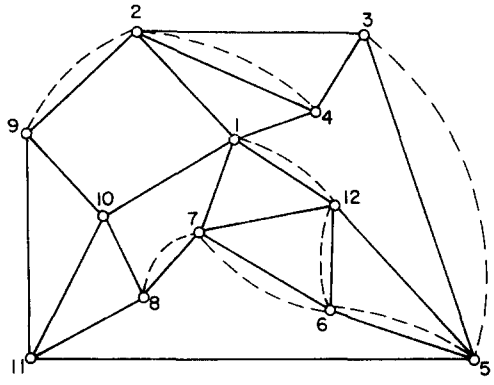



FIG. 3. *Graph $G_o$ ($M_{min}$) for example A.*
*——————— real links*
*– – – – – – – – artificial links*

The graph $G_o(M_{min})$ at the end of step 3 is, therefore, as shown in Fig. 3, where the artificial links are shown dotted. From this figure it is seen that the optimum cycle covering $G$ traverses links (9, 2) (2, 4) (3, 5) (5, 6) (1, 12) (12, 6) (7, 6) and (8, 7) twice and all other links once. The cost of this cycle is therefore (from Table 1) 294 units.

Once the graph $G_o(M_{min})$ is found, an eulerian cycle (traversing each link of this graph once and once only) and the corresponding optimum cycle covering the original graph $G$, can be constructed immediately [8]. Thus in $G_o(M_{min})$, starting from any vertex traverse and erase from the graph any incident link, unless the removal of that link will cause the graph to be separated into two disconnected components (not counting isolated single vertices). For the graph

of Fig. 3, for example, one possible eulerian cycle obtained from the above rule is given by the vertex sequence (starting from vertex 1)
1, 4, 3, 5, 3, 2, 4, 2, 1, 12, 5, 6, 7, 8, 10, 1, 7, 6, 12, 6, 5, 11, 9, 2, 9, 10, 11, 8, 7, 12, 1.

This cycle is also the corresponding optimum covering cycle of the original graph $G$ of Fig. 2. Other cycles which cover $G$ by traversing the links in a different order to the one shown above (but traversing the same set of 8 links twice) are also possible. Thus once the links of $G$ that must be traversed twice by an optimal cycle are determined, there may, in general, be more than one optimal cycle covering $G$.

## ALGORITHM FOR PROBLEM B

A heuristic algorithm for the solution of problem B will now be described which is based on the previous algorithm for problem A. The algorithm is described below with enough explanation included at each step to make any further comment unnecessary.

*Step* 1

Starting with vertex $x_1$ of graph $G$, attempt to construct a feasible cycle (i.e. a cycle having a sum of $q_{ij}$'s which is less than $W$). The cycle should be such that when its links are removed from $G$ the remaining graph should be connected. If such a cycle exists it can be constructed by sequentially forming a path $x_1, \ldots, x_i$ traversing any link $(x_i,x_j)$ which:

(i) When removed does not separate $G$ into disconnected components (not counting isolated vertices).

(ii) The total quantity (sum of the $q$'s) of the path $x_1, \ldots, x_i$ plus the total quantity of the least-quantity path from $x_j$ to $x_1$ should be less than $W$.

Condition (ii) implies that before link $(x_i,x_j)$ is added to the cycle being formed the least-quantity path from $x_j$ back to $x_1$ should be calculated—for example by the Dijkstra [4] algorithm using the quantity matrix $Q = [q_{ij}]$. However, in general it will not be necessary for such an algorithm to terminate before it can be seen that a return path (from $x_j$ to $x_1$) of sufficiently low quantity is available (or not as the case may be) to complete a feasible cycle.

If no link $(x_i,x_j)$ satisfying (i) and (ii) above can be found, then the cycle is formed by the path $x_1, \ldots, x_i$ and the least quantity return path from $x_i$ to $x_1$. (This cycle is by necessity feasible, otherwise the last link of the path $x_1, \ldots, x_i$ would never have been chosen.)

If step 1 has produced some feasible cycle the step is repeated until no more feasible cycles are produced and then the algorithm proceeds to step 2. If no feasible cycle at all has been found the algorithm proceeds to step 3.

*Step* 2

If the graph (remaining after the feasible cycles determined by step 1 are removed) contains artificial links remove these links and go to step 3. (In the first pass no artificial links will exist.)

*Step* 3

(a) If the remaining graph has vertex degrees which are all even or zero (except for vertex $x_1$ whose degree must be even but not zero) then add the two shortest paths of artificial links to the remaining graph from $x_1$ to the vertex nearest $x_1$. Return to repeat step 1, where $G$ is now the remaining graph (plus the added artificial links). If there are still no feasible cycles add two more shortest paths of artificial links from $x_1$ to its second nearest vertex, etc. until a feasible cycle is found by step 1.

(b) If the remaining graph has vertices with odd degrees (or vertex $x_1$ has degree zero) use the algorithm of problem A to find the minimal matching of odd-degree vertices using the original cost matrix $C = [c_{ij}]$. This matching produces the lowest total cost artificial links to be added to the remaining graph in order to have even vertex degrees. In the case where $x_1$ has degree zero, $x_1$ is included in the matching by considering it as a double vertex $x_1'$ and $x_1''$ with infinity as the cost of matching $x_1'$ to $x_1''$. The result would then be a matching of $x_1'$ and $x_1''$ to two other vertices, i.e. the degree of $x_1$ will become 2.

Return to repeat step 1 where the graph is now the remaining graph plus the artificial links introduced by the matching.

*Step* 4

Repeat steps 1–3 until the whole graph has been covered, i.e. there is no remaining graph at step 2.

It should be noted here that all artificial links introduced by step 3 are considered at step 1 to have zero quantities $q_{ij}$ associated with them, so that a cycle is feasible or not depending only on the total quantity of the real (non-artificial) links in it. An artificial link $(x_i,x_j)$ does of course have a cost $c_{ij}$ associated with it, which is equal to the cost of the corresponding real link in $G$ between the same two vertices.

## EXAMPLE B

Consider once more the graph of Fig. 2 with the cost matrix given in Table 1. We will assume the quantities $q_{ij}$ of all links to be unity and take $W$ to be 5 units.

In the first pass step 1 determines two feasible cycles in $G$ which leave the remaining graph connected. These two cycles are: (1, 10, 9, 2, 4, 1) and (1, 12, 5, 6, 7, 1). (Other choices of two feasible cycles also exist and might

have been the ones chosen.) Because of the simplicity of the example, these cycles can be quite easily determined by inspection without the use of any shortest-path algorithm as suggested in step 1.

The algorithm proceeds to step 2 but since the remaining graph shown in Fig. 4(a) contains no artificial links, it continues to step 3. Since the graph of

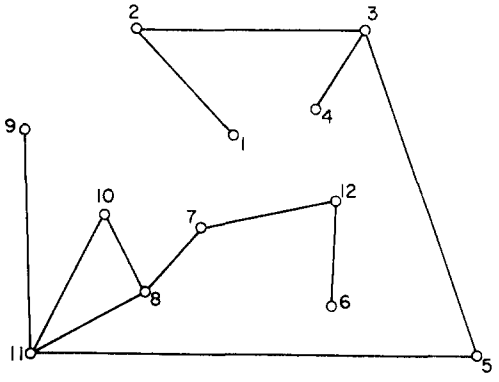



FIG. 4(a)

Fig. 4(a) possesses odd-degree vertices, case (b) applies. The odd-degree vertices to be matched are the same as for example A worked out earlier, so that the minimal matching and the paths of artificial links to be introduced into the remaining graph are as given by expression (2). The graph at the beginning of the second pass through step 1 is then as shown in Fig. 4(b).

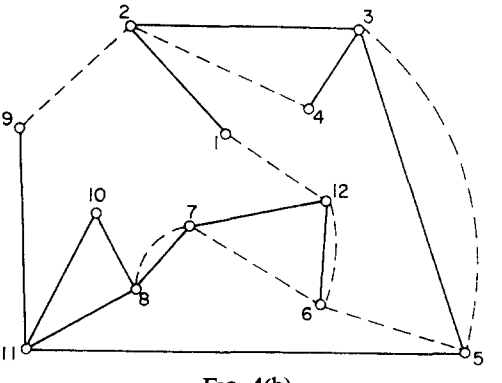



FIG. 4(b)

Step 1 now determines the feasible cycle (again in this simple case it can be determined by inspection), (1, 2, 9, 11, 10, 8, 7, 6, 12, 1), where artificial links are shown underlined. (It should be noted that although this cycle contains 9 links, 4 of them are artificial so its total quantity is $5 = W$.)

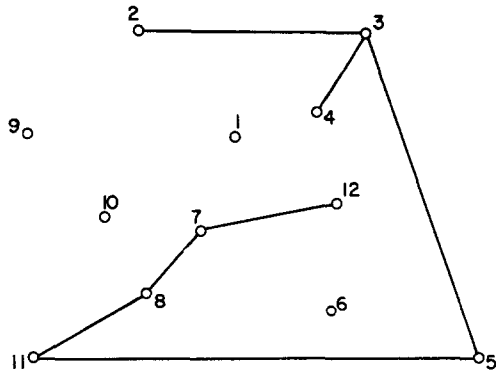



FIG. 4(c)

Step 2 now reduces the graph to that shown in Fig. 4(c) and since the graph contains odd-degree vertices the algorithm continues to step 3(b). According to this step the vertices to be matched are now: 1′, 1″, 2, 3, 4 and 12. Following the algorithm for problem A, the least cost path matrix $D = [d_{ij}]$ is found to be

$D =$

| | 1′ | 1″ | 2 | 3 | 4 | 12 |
|---|---|---|---|---|---|---|
| 1′ | X | 1″ | | | | |
| 1″ | ∞ | X | 2 | | | |
| 2 | 13 | 13 | X | 3 | | |
| 3 | 19 | 19 | 18 | X | 4 | |
| 4 | 17 | 17 | 9 | 20 | X | 12 |
| 12 | 4 | 4 | 17 | 15 | 21 | X |

and the minimum matching is then calculated to be

$$\left.\begin{array}{l} 1' \text{ to } 12, \text{ path } 1-12 \\ 1'' \text{ to } 3, \text{ path } 1-12-6-5-3 \\ 2 \text{ to } 4, \text{ path } 2-4 \end{array}\right\} \quad (3)$$

The graph at the beginning of the third pass through step 1 is, therefore, as shown in Fig. 4(d).

Step 1 now determines the feasible cycle

$$(1,\ 12,\ 7,\ 8,\ 11,\ 5,\ 3,\ 5,\ 6,\ 12,\ 1)$$

again with artificial links shown underlined.

Step 2 reduces the graph to that shown in Fig. 4(e). Step 3(b) calculates the matching:

$$\left.\begin{array}{l} 1' \text{ to } 2, \text{ path } 1\text{–}2, \\ 1'' \text{ to } 4, \text{ path } 1\text{–}4, \end{array}\right\} \quad (4)$$

so that the graph at the beginning of the fourth pass through step 1 is as shown in Fig 4(f).

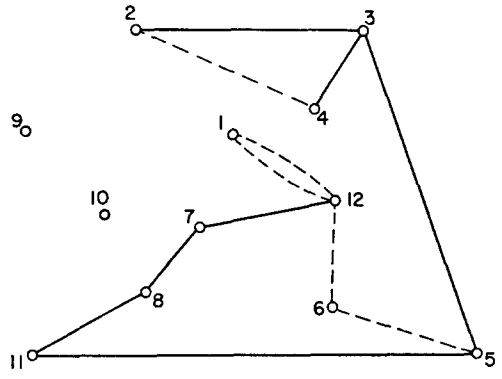
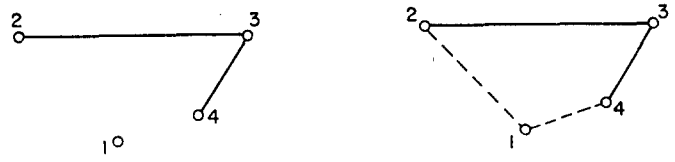



FIG. 4(d)




FIGS. 4 (e) and (f) (*The vertices not shown are isolated.*)

Step 1 now determines the final feasible cycle

$$\underline{1,\ 2},\ 3,\ \underline{4,\ 1}$$

and the algorithm terminates.

The five feasible cycles covering the original graph $G$ are therefore

| | | | |
|---|---|---|---|
| | (1, 10, 9, 2, 4, 1) | cost: | 63 |
| | (1, 12, 5, 6, 7, 1) | cost: | 45 |
| | (1, 2, 9, 11, 10, 8, 7, 6, 12, 1) | cost: | 63 |
| | (1, 12, 7, 8, 11, 5, 3, 5, 6, 12, 1) | cost: | 84 |
| and | (1, 2, 3, 4, 1) | cost: | 68 |
| | | Total cost: | 323 units |

This total cost compares with a lower bound to the solution of problem B (this is the cost of the optimal solution of problem A) of 294 units. In fact this lower bound can be increased further. Since the total number of links in the initial graph $G$ is 22, each with an assumed quantity of unity whereas the maximum capacity of a cycle was taken to be 5, at least five cycles are required to cover $G$, and hence 10 links (real and artificial) must be incident at vertex 1. Since the optimal solution to problem A requires 6 links incident at 1 (see Fig. 4) at least 4 more artificial links are required for problem B. If these links are taken as the shortest possible (i.e. links from 1 to 12) an extra cost

of at least $4 \times 4 = 16$ units is incurred by any solution to problem B. A better lower bound to this solution is therefore $294 + 16 = 310$ units.

The cost of the solution to problem B found above is only about 4 per cent higher than the lower bound. However, many attempts to improve on the solution have failed and it is believed that the solution found may in fact be the optimal solution.

## SOME COMPUTATIONAL RESULTS

Problem A was solved for five randomly generated graphs varying in size from 10 vertices and 25 links to 50 vertices and 125 links, the cost matrices $[c_{ij}]$ also being random but symmetrical. The computational results are shown in Table 2. Although the table is not extensive enough to be able to form a precise relationship between the computing times and the graph size, the relation between this time $T$ and the number of links $m$ in $G$, seems to be a low order monomial namely $T$ is proportional to $m^k$ where $k = 2$ or 3.

TABLE 2. COMPUTATIONAL RESULTS FOR PROBLEM A

| Graph | | Computing time ($T$) |
|---|---|---|
| Number of vertices ($n$) | Number of links ($m$) | CDC 6600 sec |
| 10 | 25 | 0·21 |
| 20 | 50 | 2·36 |
| 30 | 75 | 11·0 |
| 40 | 100 | 9·25 |
| 50 | 125 | 16·6 |

Problem B was solved for the same set of graphs as above, the link quantities $q_{ij}$ being randomly generated with uniform distribution in the range 10–30, and with the value of $W$ taken as 100. The results are shown in Table 3 together with the lower bounds calculated from the solution of problem A for those graphs. It is seen from this table that the solution values are, on average, about 5·6 per cent above the lower bound. The values are, however, believed to be even closer to the true optimal solutions.

TABLE 3. COMPUTATIONAL RESULTS FOR PROBLEM B

| Graph | | Computing time $T$ (sec) | Value of solution | Lower bound | % value above lower bound |
|---|---|---|---|---|---|
| $n$ | $m$ | | | | |
| 10 | 25 | 0·57 | 537 | 511 | 5·09 |
| 20 | 50 | 4·75 | 1343 | 1296 | 3·63 |
| 30 | 75 | 21·2 | 1599 | 1467 | 9·00 |
| 40 | 100 | 18·4 | 2674 | 2563 | 4·33 |
| 50 | 125 | 28·4 | 2862 | 2701 | 5·96 |

## REFERENCES

1. BELLMAN R and COOKE KL (1969) The Konigsberg bridges problem generalised. *J. math. Analysis Applic.*, **25,** 1.
2. CHRISTOFIDES N and EILON S (1969) An algorithm for the vehicle dispatching problem. *Opl Res. Q.*, **20,** 309.
3. CLARKE G and WRIGHT JW (1963) Scheduling of vehicles from a central depot to a number of delivery points. *Ops Res.*, **11,** 568.
4. DIJKSTRA EW (1959) A note on two problems in connection with graphs. *Num. Math.*, **1,** 269.
5. EDMONDS J (1965) Maximum matching and a polyhedron with 0–1 vertices. *J. Res. natn. Bur. Stand.*, **69B,** 125.
6. EDMONDS J (1965) The Chinese postman's problem. *ORSA Bull.*, **13,** 73.
7. EILON S, WATSON-GANDY CDT and CHRISTOFIDES N (1971) *Distribution Management.* Griffin, London.
8. KAUFMANN A (1967) *Graphs, Dynamic Programming and Finite Games*, p. 309. Academic Press, New York.
9. MEI-KO KWAN (1962) Graphic programming using odd or even points. *Chinese Math.*, **1,** 273.