Approximation Methods for Finding a Hamiltonian Walk with Minimum Cost

Approximation Methods for Finding a Hamiltonian Walk with Minimum Cost

Syoichiro YAMADA*, Akira UMEZU**, Tetsuo OHNO*** and Tamotsu KASAI*

(Received June 15, 1982)

This paper describes two types of approximation methods for finding a hamiltonian walk with minimum cost in a given graph. One is the constructive method, and the ratio of the cost of the solution to that of the optimum solution has the worst-case bound of 2. The other is the iterative improvement method which is based on the well-known λ-optimal method. The computation times of them are proportional to $O(n^2+m\log m)$, where n and m are the numbers of vertices and edges of the graph, respectively.

1. Introduction

The problem for finding a hamiltonian walk with minimum cost in a given graph is closely related to the well-known travelling salesman problem, and solutions to this problem and its variants have practical applications in many fields.

This problem is described as follows: Given a connected graph G, find a closed walk with minimum cost, which passes through every vertex of G *at least once*. Note that this problem is a generalization of the travelling salesman problem which is defined as follows: Given a connected graph G, find a circuit with minimum cost, which passes through every vertex of G *exactly once*. Since this problem is one of the NP-hard problems,[1] it is not reasonable to expect to find a polynomial-time algorithm for its exact solution. Therefore, it has been desired to develop an efficient technique that yields a good approximate solution to this problem. However, reports of such research have apparently not been published to date except for the case of graphs without costs.[2] We can immediately replace this problem with the travelling salesman problem, by finding the shortest paths between any two vertices of the graph. In this case the time complexity is $O(n^3)$, where n is the number of vertices of the graph. Since it is generally said that the desired time complexity of algorithms for applying to large networks or systems is $O(n)$ or at most $O(n^2)$, the efficiency of this method will not be satisfactory for practical applications.

In this paper we propose two types of approximation methods with time complexity $O(n^2+m\log m)$, where m is the number of edges of the graph, for finding a hamiltonian walk with minimum cost in a given graph, and discuss the accuracy and average computation times of them. One is a constructive method based on the shortest spanning tree, and the ratio of the cost of the solution to that of the optimum solution has the worst-case bound of 2. The other method is based on the λ-optimal

* Department of Electrical Engineering, College of Engineering.
** Nippon Electric Co., Ltd.
*** Graduate Student, Department of Electrical Engineering, College of Engineering.

method[3] which is an efficient heuristic method to solve the combinatorial optimization problems.

2. Preliminaries

This chapter introduces some of concepts and terms used in this paper. We deal with a connected and undirected graph with positive cost written along the edges. A *hamiltonian walk* is a closed walk which passes through every vertex of the graph G at least once. A *cactus* is defined as a connected graph in which every block is either a bridge or a circuit without edges that join two non-consecutive vertices of the circuit. Hence, in the cactus every pair of circuits, which is adjacent each other, always has a common vertex. A graph $H(G_r)$ is defined as a graph which represents a hamiltonian walk of a graph G_r, i.e., the graph $H(G_r)$ means a multigraph, in which every bridge of G_r has a pair of edges in parallel.

Throughout this paper n and m denote the numbers of vertices and edges of the graph G, respectively. The symbol $c(i, j)$ is used to represent the cost of an edge (i, j), similarly $c(G_r)$ is the sum of costs of edges in G_r. Hereafter, we will call $c(G_r)$ simply the cost of G_r. Furthermore, the cost of a hamiltonian walk is written by h, and its minimum value is h_m.

3. Constructive Method

Let T be a spanning tree of a graph G. Then we can consider a hamiltonian walk of G, which can be constructed by traversing twice every edge of T. Therefore denoting a shortest spanning tree of the graph G by T_s, the following inequality is derived:

$$c(T_s) < h_m \leqq 2 \times c(T_s). \tag{1}$$

Now let us assume that there exists a co-tree of T_s, and the cost of its edge (i, j) is less than that of a path $P_s(i, j)$ in T_s, i.e., $c(i, j) < c(P_s(i, j))$. Then adding an edge (i, j) to T_s and forming a circuit, a new hamiltonian walk, whose cost is less than $2 \times c(T_s)$, can be constructed. Hence, by repeating above operations, it will be expected that we can find a hamiltonian walk whose cost is approximately equal to h_m.

On the basis of the above considerations, an approximation algorithm to solve the problem for finding a hamiltonian walk with minimum cost is deduced as follows:

[Algorithm 1]

Step 1: Find a shortest spanning tree T_s.
Let T_c be a co-tree of T_s.
Set $H_1 \leftarrow T_s$, $H_2 \leftarrow \phi$, and $h \leftarrow 2 \times c(T_s)$.

Step 2: Calculate a distance matrix D_s of T_s.

Step 3: Let $\varDelta h(i, j)$ be a decreased value of h by adding an edge (i, j) to the graph H_1, i.e., $\varDelta h(i, j) = D_s(i, j) - c(i, j)$.
Calculate $\varDelta h(i, j)$ for every edge $(i, j) \in T_c$.

Step 4: Remove a set of edges, $\{(i, j)\}$ with $\varDelta h(i, j) \leqq 0$, from T_c. Arrange the edges belonging to T_c in the non-increasing order of $\varDelta h$.

Step 5: If T_c becomes empty, then go to Step 6. Otherwise choose the first element (i, j) of T_c, and set

$$T_c \leftarrow T_c - \{(i, j)\}.$$

Find a path $P(i, j)$ in H_1. If there is no path between the vertices i and j, repeat Step 5. Otherwise set

$$H_1 \leftarrow H_1 - P(i, j),$$

$$H_2 \leftarrow H_2 \cup \{(i, j)\} \cup P(i, j),$$

and

$$h \leftarrow h - \Delta h(i, j),$$

and repeat Step 5.

Step 6: Add an edge in parallel with every edge in H_1, and set

$$H(G) \leftarrow H_1 \cup H_2,$$

and stop.

We shall show that the time complexity of this algorithm is $O(n^2 + m\log m)$. It is clear that Steps 1 and 2 require $O(n^2)$ operations. Since the number of edges in T_c is $m-n+1$, Step 3 is executed $m-n+1$ times, and Step 4 requires at most $O(m\log m)$ operations. Step 5 is repeated until it becomes impossible to find any path $P(i, j)$ in H_1. Therefore the total amount of operations required at Step 5 is $O(m)$. Step 6 clearly needs $O(m)$ operations. Thus algorithm 1 requires $O(n^2 + m\log m)$ total operations.

[Example 1]

Let G be the graph depicted in Fig. 1. We show the procedure to find the hamiltonian walk in G by using the algorithm 1.

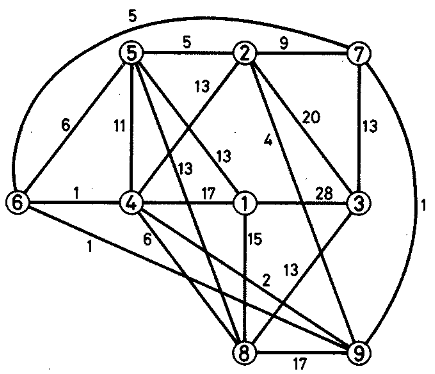

Fig. 1 A graph G.

The cost matrix $[c(i,j)]$ of G is given by Eq. (2):

$$[c(i,j)]=\begin{pmatrix} - & - & 28 & 17 & 13 & - & - & 15 & - \\ - & - & 20 & 13 & 5 & - & 9 & - & 4 \\ 28 & 20 & - & - & - & - & 13 & 13 & - \\ 17 & 13 & - & - & 11 & 1 & - & 6 & 2 \\ 13 & 5 & - & 11 & - & 6 & - & 13 & - \\ - & - & - & 1 & 6 & - & 5 & - & 1 \\ - & 9 & 13 & - & - & 5 & - & - & 1 \\ 15 & - & 13 & 6 & 13 & - & - & - & 17 \\ - & 4 & - & 2 & - & 1 & 1 & 17 & - \end{pmatrix}. \tag{2}$$

The shortest spanning tree T_s obtained in Step 1 is shown in Fig. 2. Set $H_1 \leftarrow T_s$ and $H_2 \leftarrow \phi$.

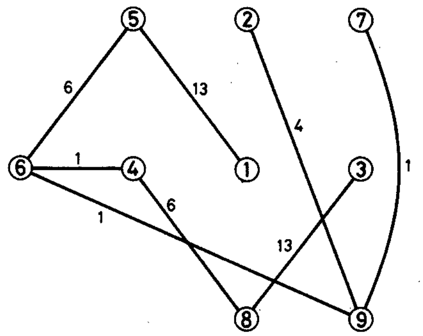

Fig. 2 A shortest spanning tree T_s ($h=2\times c(T_s)=90$).

The distance matrix $[D_s(i,j)]$ of T_s is obtained as Eq. (3):

$$[D_s(i,j)]=\begin{pmatrix} - & 24 & 39 & 20 & 13 & 19 & 21 & 26 & 20 \\ 24 & - & 25 & 6 & 11 & 5 & 5 & 12 & 4 \\ 39 & 25 & - & 19 & 26 & 20 & 22 & 13 & 21 \\ 20 & 6 & 19 & - & 7 & 1 & 3 & 6 & 2 \\ 13 & 11 & 26 & 7 & - & 6 & 8 & 13 & 7 \\ 19 & 5 & 20 & 1 & 6 & - & 2 & 7 & 1 \\ 21 & 5 & 22 & 3 & 8 & 2 & - & 9 & 1 \\ 26 & 12 & 13 & 6 & 13 & 7 & 9 & - & 8 \\ 20 & 4 & 21 & 2 & 7 & 1 & 1 & 8 & - \end{pmatrix}. \tag{3}$$

The values of $\Delta h(i,j)$ calculated in Steps 3 and 4 become as follows:

$\Delta h(1, 3)=11$,
$\Delta h(1, 8)=11$,
$\Delta h(3, 7)=9$,
$\Delta h(2, 5)=6$,
$\Delta h(2, 3)=5$,
$\Delta h(1, 4)=3$.

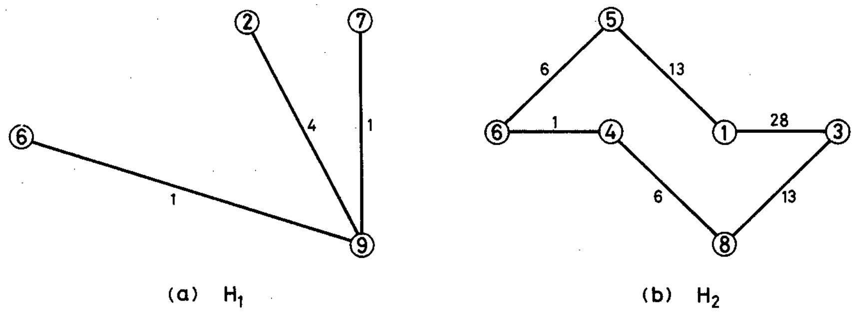

Fig. 3 H_1 and H_2 (h=79).

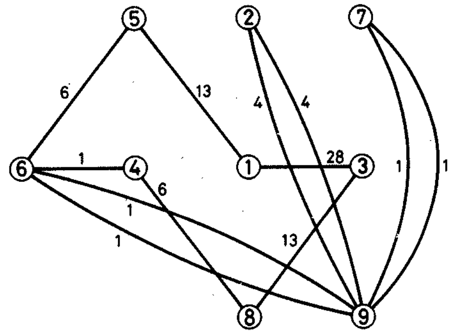

Fig. 4 A graph $H(G)$.

Then the first element of T_c is the edge (1, 3). Finding the path P(1, 3) of $H_1(=T_s)$, we have $P(1, 3)=\{(1, 5), (5, 6), (6, 4), (4, 8), (8, 3)\}$. The resultant graphs H_1 and H_2 are shown in Fig. 3.

As it is impossible to find any more path $P(i, j)$, where $(i, j)\in T_c$, we cannot improve the solution, hence go to Step 6. A graph $H(G)$ obtained in Step 6 is shown in Fig. 4. Thus the algorithm finds a hamiltonian walk of cost 79 in G.

As seen from this example, $H(G)$ obtained by the algorithm 1 is a connected graph constructed by fundamental circuits and circuits of length 2. In other words, $H(G)$ is a cactus in which every bridge has two edges in parallel. Furthermore it is clear that the cost of $H(G)$ obtained by the algorithm 1 is at most $2\times h_m$.

4. Iterative Improvement Method

The λ-optimal method has been known as one of the most effective methods for solving the combinatorial optimization problems in the graph theory. In this method the solution is iteratively improved by exchanging λ edges contained in an initial graph H_0 for another λ edges which are not in H_0. In order to restrict the computation time in practical range, we use 2-optimal method with two conditions as follows:

1) An initial graph H_0 is a cactus in which every bridge has two edges in parallel.
2) The patterns obtained by exchanging two pairs of edges are restricted only three

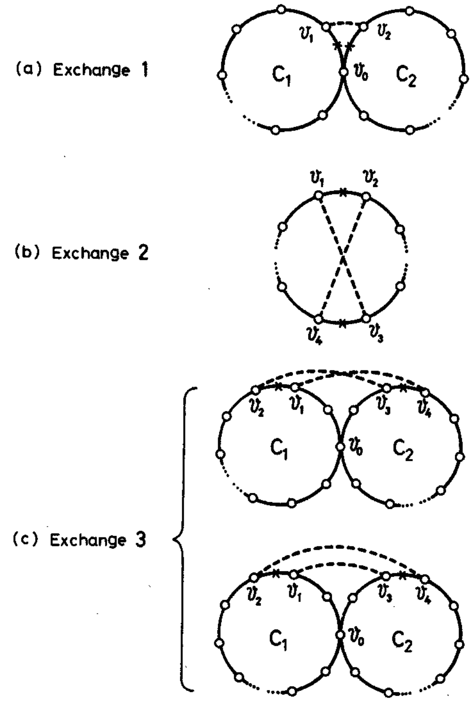

Fig. 5 2-optimal method.

types shown in Fig. 5, where broken lines represent the edges to be added and the lines marked with crosses represent the edges to be removed to improve the solution. We will call these operations, shown in Fig. 5(a), (b) and (c), exchange 1, 2 and 3, respectively.

Exchange 1 intends the improvement of the solution by replacing a pair of adjacent circuits with a circuit. As shown in Fig. 5(a), let C_1 and C_2 denote adjacent circuits with a common vertex v_0, and v_1 and v_2 denote the vertices adjacent to v_0. Then this operation is described as the following. If and only if there exists an edge (v_1, v_2) in G and the inequality denoted by Eq. (4) is satisfied, the edges (v_0, v_1) and (v_0, v_2) are removed, and the edge (v_1, v_2) is added:

$$c(v_1, v_2) < c(v_0, v_1) + c(v_0, v_2). \tag{4}$$

Exchange 2 intends the improvement of the solution by exchanging a pair of non-consecutive edges in a circuit for a pair of edges which are not contained in the circuit. As shown in Fig. 5(b), if and only if there exists a pair of edges (v_1, v_3) and (v_2, v_4), and the inequality denoted by Eq. (5) is satisfied, by removing the edges (v_1, v_2) and (v_3, v_4), and by adding the edges (v_1, v_3) and (v_2, v_4), the solution can be improved:

$$c(v_1, v_3) + c(v_2, v_4) < c(v_1, v_2) + c(v_3, v_4). \tag{5}$$

Similarly, exchange 3 intends the improvement of the solution by exchanging a pair of edges for another pair of edges of adjacent circuits. As shown in Fig. 5(c), if and only if edges (v_1, v_4) and (v_2, v_3) or edges (v_1, v_3) and (v_2, v_4) are contained in G, and the inequality denoted by Eq. (6) or (7) is satisfied, by removing the edges (v_1, v_2) and (v_3, v_4), and by adding the edges (v_1, v_4) and (v_2, v_3) or the edges (v_1, v_3) and (v_2, v_4), the solution can be improved:

$$c(v_1, v_4)+c(v_2, v_3)<c(v_1, v_2)+c(v_3, v_4), \tag{6}$$

$$c(v_1, v_3)+c(v_2, v_4)<c(v_1, v_2)+c(v_3, v_4). \tag{7}$$

Note that the walk obtained by our method is guaranteed to be connected by setting two conditions described before.

The outline of the algorithm will be described below.

[Algorithm 2]

Step 1: Set $H_w \leftarrow H_0$.

Step 2: Apply exchange 1 to every pair of adjacent circuits in H_w.

Step 3: Apply exchange 2 to all circuits in H_w.

Step 4: Apply exchange 3 to every pair of adjacent circuits in H_w.

Step 5: If the value of h is decreased by the operations of Steps 2, 3 and 4, repeat Step 2, otherwise stop.

It is clear that Steps 2, 3 and 4 are executed in $O(n^2)$ operations. Since the process of repeating Steps 2, 3 and 4 will terminate when and only when the solution has no room of the improvement, the time complexity of the algorithm 2 is $O(Kn^2)$, where K is the number of the iterations.

[Example 2]

As an example, let us consider again the graph G shown in Fig. 1. Suppose we choose H_0 shown in Fig. 6, as an initial graph. The graphs obtained by applying the algorithm 2 are shown by solid lines in Figs. 7, 8 and 9. The value of h has been improved from 95 to 68. Since the minimum value of h is 68, our method also

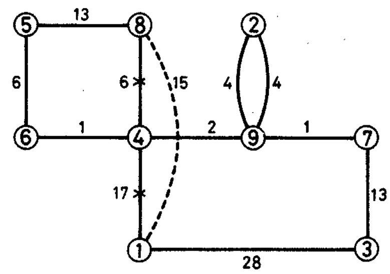

Fig. 6 Initial graph H_0 (h=95).

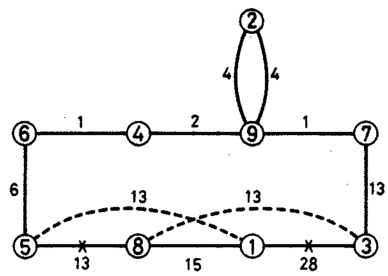

Fig. 7 Improved result by exchange 1 (h=87).

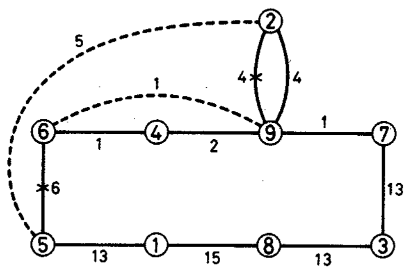

Fig. 8 Improved result by exchange 2 (h=72).

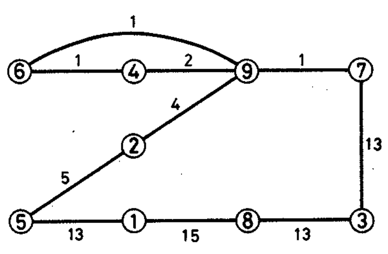

Fig. 9 Improved result by exchange 3 (h=68).

produces the optimum solution in this example.

Two iterative improvement methods can be considered according to how to choose the initial graph in algorithm 2. In the first method the initial graph is constructed by using algorithm 1. It will be clear that the initial graph obtained by this method satisfies the condition 1) mentioned previously. Hereafter we will call it simply the iterative method I. In the second method the initial graph is constructed, based on the shortest-distance tree with an appropriate starting vertex, by the same technique as the constructive method. We will call the second method simply the iterative method II. Since the iterative method II is permitted to choose many distinct initial graphs, it will be expected to get over some local minima.

5. Experimental Results

The ratios between the values of optimum and suboptimum solutions are listed in Table 1. These data were obtained on random graphs with uniformly distributed random costs, and the number of the data is 20 for $n=5$ and 10, and 7 for $n=15$. We obtained the optimum solutions by using the branch-and-bound method.[4] Since it is extremely difficult to find the optimum solutions in large-scale problems, the solutions in only small problems were shown here.

It was described before that the ratio R for the constructive method is at worst 2, however, from this results, $R_a<1.1$ for $n<15$, i.e., the average computational error is less than 10 percents for $n<15$. This fact means that such a method has high accuracy for small size graphs. In the iterative method II on over 20 trials with different initial graphs, the computational error is decreased in about 5 percents as compared with that of the constructive method.

Figure 10 illustrates the average computation time t for the graph with $m=5n$ edges, where the program was written in FORTRAN and run on NEAC ACOS-77/700. In this figure it is confirmed that the average computation time of every method is $O(n^2)$, if the graphs used as data are sparse, that is, the number of edges are far fewer than $n(n-1)/2$. Furthermore it became clear that the average value of the repetition number K in the iterative method II was constant independently of n.

Table 1 The ratio R between the values of the optimum and suboptimum solutions

n		5	10	15
Constructive method	R_a	1.022	1.072	1.103
	R_m	1.146	1.247	1.181
Iterative method I	R_a	1.009	1.052	1.079
	R_m	1.117	1.220	1.168
Iterative method II	R_a	1	1.029	1.056
	R_m	1	1.152	1.124

R_a and R_m: average and maximum values of R.

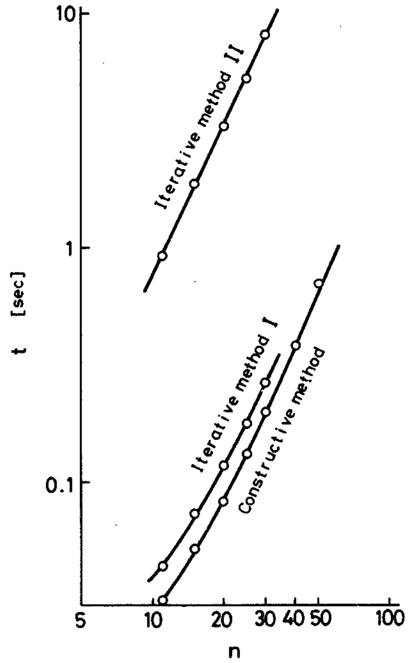

Fig. 10 Computation time t.

6. Conclusions

We proposed in this paper two types of approximation methods, i.e., the constructive method and the iterative improvement method, for finding the hamiltonian walk with minimum cost, and discussed them from the view point of the accuracy and computation time. The results are summarized as follows:

1) The ratio of the cost of the solution obtained by the constructive method to that of the optimum solution has the worst-case bound of 2.

2) The average computational error of the constructive method is less than 10 percents in the range of $n<15$.

3) The average computational error of the iterative improvement method with random choice of initial graphs can be decreased in about 5 percents as compared with that of the constructive method, when we select 20 for the number of iterations. Although by increasing the number of iterations the accuracy may be higher, the computation time will increase rapidly.

4) The average computation time of our method is proportional to n^2, when the given graphs are so sparse that the ratio m/n is constant.

Since almost all graphs representing large networks or systems are sparse, it can be understood that our methods have higher computational efficiency than the conventional methods.

References

1) A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, p. 364, Addison-Wesley, Reading, MA (1974).

2) K. Takamizawa, T. Nishizeki, and N. Saito, Networks, **10**, 249 (1980).
3) S. Lin and B. W. Kernighan, Opns. Res., **21**, 498 (1973).
4) N. Christofides, Graph Theory: An Algorithmic Approach, p. 236, Academic Press (1975).