Chapter 7: Path Algorithms

7.1 Solving Chinese Postman Problem

The Chinese Postman Problem leads us to find an optimal tour on a given connected weighted graph G. This means that we are going to find, by duplicating edges if necessary, a weighted Eulerian supergraph G^* of G such that the weight of G^* is as small as possible. Suppose $\{v_1, \ldots, v_n\}$ is the set of all odd vertices in G, where n must be even. The method is described as follows:

1. Find the distance of each pair of odd vertices of G.
2. Form a weighted complete graph K_n with vertex set $\{v_1, \ldots, v_n\}$, and assign the weight of the edge v_iv_j to be the distance $\partial(v_i, v_j)$ (which is introduced in the next section).
3. Find an optimal perfect matching of K_n, i.e., a set M of edges of K_n such that every vertex is incident with one edge of M and the sum of the weights in M is minimum.
4. An edge v_iv_j in M corresponds to a shortest (v_i, v_j)-path added into G.

Since the method involves finding matching from a weighted graph that we will not present in this book. We use brute force method to find an optimal perfect matching in K_n for small n.

Example 7.1.1: Let the vertex set of K_4 be $\{A, B, C, D\}$ and the weight of each edge is shown in the following table:

	A	B	C	D
A	0	3	2	2
B	3	0	4	1
C	2	4	0	4
D	2	1	4	0

There are only three perfect matchings of K_4, namely $M_1 = \{AB, CD\}$, $M_2 = \{AC, BD\}$ and $M_3 = \{AD, BC\}$. Their weights are $W(M_1) = W(AB) + W(CD) = 7$, $W(M_2) = 3$ and $W(M_3) = 6$. So the optimal matching is M_2.

7.2 Shortest Path Algorithm

Let u and v be vertices of G (which is a graph or a digraph). The *length* of a (u, v)-path P is the sum of weights of edges in P. The *distance* from u to v, denoted by $\partial(u, v)$, is the minimum length of all (u, v)-paths in G. If there is no (u, v)-path, then we define $\partial(u, v) = \infty$. Note that $\partial(u, v)$ and $\partial(v, u)$ may not be the same in a digraph. In particular, if all the weights are 1, then $\partial(u, v) = d(u, v)$. In this chapter, we assume that all graphs (or digraphs) are connected.

Given a weighted graph or digraph with a nonnegative weight function W. We want to find a shortest path from a vertex s to another vertex t. Note that the shortest path may not unique, but of course the distance is unique.

Idea: Each vertex v is assigned an ordered pair $(\lambda(v), p(v))$, where $\lambda(v)$ represents the *tentative shortest distance* from s to v by all paths considered until now, and $p(v)$ denotes the predecessor of v in the tentative shortest (s, v)-path. The vertex at any iteration that yields the smallest $\lambda(v)$ is termed its

permanent distance, and denoted by a square around the ordered pair in which $\lambda(v)$ represents the permanent distance from s to that vertex. Suppose u has been assigned the permanent distance, then $\lambda(u)$ is the actual distance $\partial(s,u)$ from s to u. Once t has been assigned the permanent distance, then the distance from s to t is determined. We use uv to denote the edge $\{u,v\}$ or the arc (u,v) in the following.

Vertex-labeling Algorithm

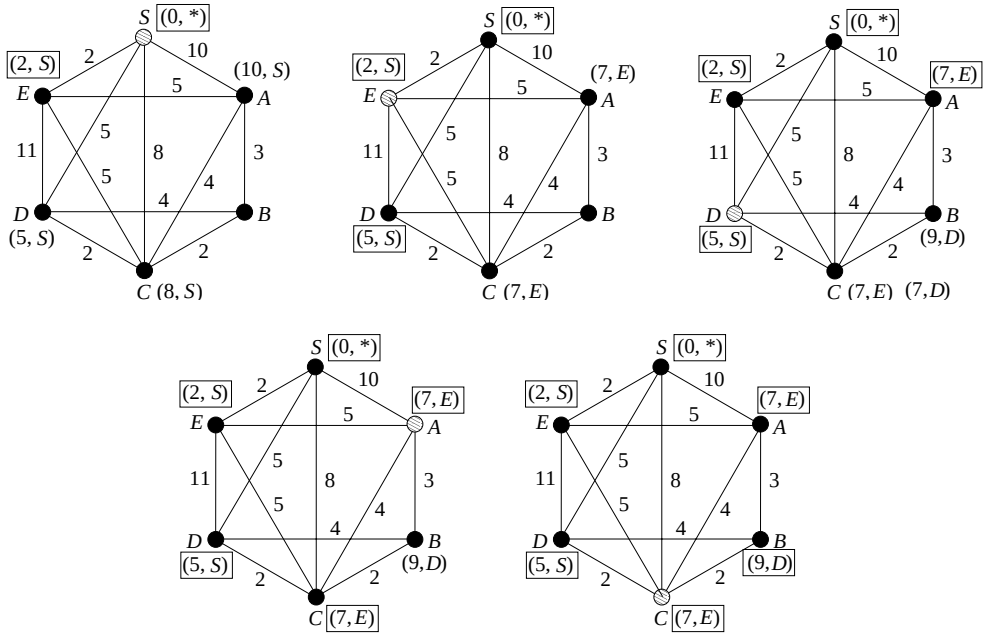
Step 1. Assign the ordered pair $(0,*)$ to vertex s; label each vertex v that is adjacent to s with the distance which is equal to the weight of sv and the predecessor is s; choose the smallest value among these labels and mark the permanent distance of the corresponding vertex. Draw a rectangle around the corresponding ordered pair.

Step 2. Consider the vertex u that just marked the permanent distance; look at each vertex v that is adjacent to u and assign the label $(\partial(s,u)+W(uv),\ u)$ to v unless v persist a label of smaller value; when all such vertices v are labeled, choose the smallest label in the graph that does not have permanent distance and mark it as the permanent distance where it occurs; draw a rectangle around the corresponding ordered pair.

Step 3. If t has been assigned the permanent distance, then this is $\partial(s,t)$; go to Step 4, otherwise go to Step 2.

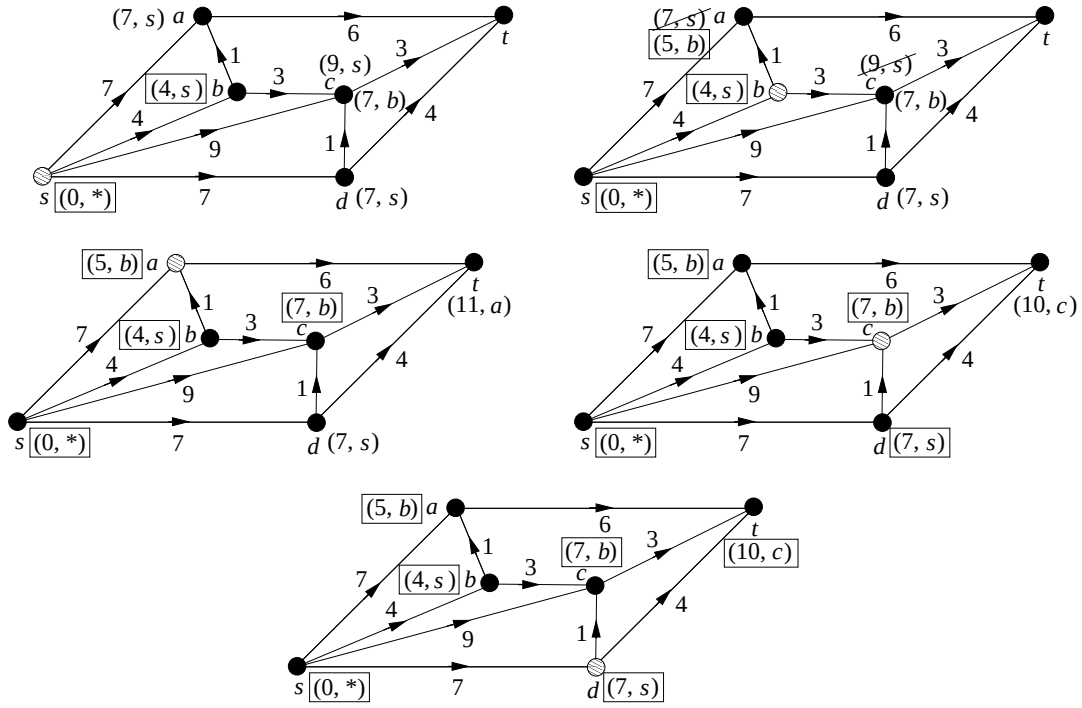
Step 4. Backtracking the predecessor vertices one by one from t.

Example 7.2.1: Consider the following weighted graph. We want to find the corresponding shortest path from S to all other vertices.



Therefore, a shortest path from S to B is SDB with distance 9 and a shortest path from S to C is SEC with distance 7.

Example 7.2.2: Consider the following network (weighted digraph). We want to find a shortest path from s to t.



A shortest path from s to t is *sbct* with distance 10.

The next algorithm is called the Dijkstra's algorithm.

Given a weighted graph (or digraph) G and a fixed vertex u. Suppose $Z \subset V(G)$ such that $u \in Z$, and let $Z' = V(G) \setminus Z$. Define the distance $\partial(u, Z')$ from u to Z' by $\partial(u, Z') = \min\{\partial(u, z) \mid z \in Z'\}$. Let $\partial(u, Z') = \partial(u, v)$ for some $v \in Z'$. Suppose $P = uu_1 \cdots u_n v$ is a shortest (u, v)-path, where $n \geq 0$. When $n = 0$, which means $u_1, \ldots, u_n$ do not appear. Since the weight function is positive, it is clear that $u_n \in Z$ and $uu_1 \cdots u_n$ is a shortest (u, u_n)-path. So

$$\partial(u, Z') = \min\{\partial(u, x) + W(xz) \mid x \in Z, z \in Z', xz \in E(G)\}.$$

Finally, if this minimum is attained when $x = u_j$ and $z = v$, then

$$\partial(u, v) = \partial(u, u_j) + W(u_j v). \tag{1}$$

Again, given a weighted graph or digraph of order p with a weight function W. We want to find a shortest path from a vertex u_0 to other vertices.

Dijkstra's Algorithm

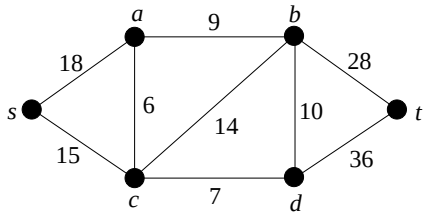
Step 1. Assign $\lambda(u_0) = 0$, $\lambda(v) = \infty$ for $v \neq u_0$, $i = 0$ and $Z_0 = \{u_0\}$. Here i denotes the index of iteration.

Step 2. For each $v \in Z_i'$, replace $\lambda(v)$ by $\min\{\lambda(v), \lambda(u_i) + W(u_i v)\}$. Compute $\min\limits_{v \in Z_i'}\{\lambda(v)\}$ and let u_{i+1} be a vertex for which this minimum is attained. Backtracking to find the first appearance of this minimum. If it appears at the j-th iteration, then define the predecessor as u_j. Assign $Z_{i+1} = Z_i \cup \{u_{i+1}\}$.

Step 3. Replace i by $i + 1$. If $i = p - 1$, then stop. If $i < p - 1$, then go to Step 2.

Theorem 7.2.3: *Suppose G is a weighted graph (or digraph) of order p. For a fixed vertex s, Dijkstra's Algorithm computes $\partial(s, v)$ for every $v \in V(G)$.*

Example 7.2.4: Consider the following weighted graph.



The following is the tabular format of Dijkstra's algorithm.

		$\lambda(\cdot)$						Remark
i	Z_i	s	a	b	c	d	t	Predecessor
		$\boxed{0}$	∞	∞	∞	∞	∞	initial
0	s		(18)	(∞)	(15)	(∞)	(∞)	$\lambda(s)+W(s\,\cdot)$
			18	∞	$\boxed{15}$	∞	∞	s
1	s,c		(21)	(29)		(22)	(∞)	$\lambda(c)+W(c\,\cdot)$
			$\boxed{18}$	29		22	∞	s
2	s,c,a			(27)		(∞)	(∞)	$\lambda(a)+W(a\,\cdot)$
				27		$\boxed{22}$	∞	c
3	s,c,a,d			(32)			(58)	$\lambda(d)+W(d\,\cdot)$
				$\boxed{27}$			58	a
4	s,c,a,d,b						(55)	$\lambda(b)+W(b\,\cdot)$
							$\boxed{55}$	b

Hence a shortest path from s to t is $sabt$ with distance 55.

Example 7.2.5: Consider Example 7.2.2 again.

		$\lambda(\cdot)$						Remark
i	Z_i	s	a	b	c	d	t	Predecessor
		$\boxed{0}$	∞	∞	∞	∞	∞	initial
0	s		(7)	(4)	(9)	(7)	(∞)	$\lambda(s)+W(s\,\cdot)$
			7	$\boxed{4}$	9	7	∞	s
1	s,b		(5)		(7)	(∞)	(∞)	$\lambda(b)+W(b\,\cdot)$
			$\boxed{5}$		7	7	∞	b
2	s,b,a				(∞)	(∞)	(11)	$\lambda(a)+W(a\,\cdot)$
					$\boxed{7}$	7	11	b
3	s,b,a,c					(∞)	(10)	$\lambda(c)+W(c\,\cdot)$
						$\boxed{7}$	10	s
4	s,b,a,c,d						(11)	$\lambda(d)+W(d\,\cdot)$
							$\boxed{10}$	c

Hence a shortest from s to t is $sbct$ of distance 10; that from s to d is sd of distance 7, etc.

7.3 Traveling Salesman Problem

The traveling salesman problem also form a *minimum-cost Hamiltonian cycle* (*optimal cycle*) in a weighted complete graph K_p. Suppose $V(K_p) = \{1, 2, \ldots, p\}$. Let $c_{i,j}$ (or c_{ij}) is the weight of the edge (or arc) ij. By convention we assign $c_{ii} = 0$ or ∞. Let C be the $p \times p$ matrix whose (i, j)-th entry is c_{ij}, which is called a *cost matrix* of the weighted graph. Minimum-cost means minimizing the sum of the cost of the edges used.

Remark 7.3.1: The concept of cost matrix can be extended to weighted non-complete graph or weighted digraph. In this situation, assign $c_{ij} = \infty$ if ij is not an edge or an arc, respectively.

There is no efficient algorithm to find an optimal solution for the traveling salesman problem. By brute force method, we have to check $(p-1)!/2$ Hamiltonian cycles for undirected complete graph K_p, which is a huge number for moderate p. In this section, we introduce a method to find an optimal solution or a near optimal solution.

7.3.1 Branch and Bound Method

The first method is a tree enumeration approach that uses a *branch and bound method* to limit our search. In this method, at most 2^p bounds should be checked, which is much less than $(p-1)!$.

To illustrate this method we consider a small traveling salesman problem with 4 vertices w, x, y, z only. Let the cost matrix C for the problem be:

	w	x	y	z
w	∞	3	9	7
x	3	∞	6	5
y	5	6	∞	6
z	9	7	4	∞

Note again, we do not require $c_{ij} = c_{ji}$; the ∞'s indicate that we cannot use these entries.

A Hamiltonian cycle contains four entries in C, one in each row and in each column, and no proper subset of the entries (edges) forms a cycle. The last constraint means that if we choose entry c_{ij}, then we cannot choose c_{ji} since these two entries from a cycle of length 2. Similarly, if entries c_{ij} and c_{jk} are chosen, then c_{ki} cannot be chosen.

Lower bound: We first demonstrate how to obtain a lower bound on the cost of this problem. Let H be a minimum-cost Hamiltonian cycle. Since H contain an entry in the first row (we will call it Row w), it will not change if we subtract a constant value from Row w of C. Note that, others rows and columns are named by their corresponding vertices. We then subtract the value of the smallest entry in Row w, namely 3, that will not create any negative entry. Repeat this procedure for other rows and we obtain a new cost matrix:

$C_1 =$

	w	x	y	z		
w	∞	0	6	4	$\to$	3
x	0	∞	3	2	$\to$	3
y	0	1	∞	1	$\to$	5
z	5	3	0	∞	$\to$	4

$L.B. = 15$

Totally we subtract $3+3+5+4 = 15$ from different rows. So a minimum-cost Hamiltonian cycle for C_1 will cost 15 less than for C. But the minimum-cost Hamiltonian cycle is still H. Similarly, we repeat the same procedure for columns and obtain

$C_2 =$

	w	x	y	z
w	∞	0	6	3
x	0	∞	3	1
y	0	1	∞	0
z	5	3	0	∞

$L.B. = 16$

The cost of H has been reduced by a total of $15 + 1 = 16$ from the cost in C. Since a minimum-cost Hamiltonian cycle for C_2 must cost at least 0, we thus obtain a lower bound on the cost of H.

Branch:

We are now ready for the part of branching. Consider an entry in C_2 that is zero, say c_{wx}. There are only two cases: the edge corresponding to c_{wx} belongs to H or not. We branch on this choice.

Suppose we do not use c_{wx}, that is, the arc (w, x) is not in H. We represent the no-(w, x) choice by setting $c_{wx} = \infty$. The smallest value in Row w of the altered C_2 is now $c_{wz} = 3$, so we can subtract this amount from Row w. Similarly we can subtract 1 from Column x and obtain the cost matrix

$$C_3 = \begin{array}{c|cccc} & w & x & y & z \\ \hline w & \infty & \infty & 3 & 0 \\ x & 0 & \infty & 3 & 1 \\ y & 0 & 0 & \infty & 0 \\ z & 5 & 2 & 0 & \infty \end{array} \qquad L.B. = 20$$

Hence the lower bound on the cost of H is 20 if H does not contain (w, x).

Suppose the arc (w, x) is in H. There is no immediate increase in lower bound on the cost of H. We now consider partial tours using c_{wx} and continue to extend these partial tours until we obtain a lower bound larger than 20. If the lower bound exceeds 20, then we consider partial tours not using c_{wx}.

Our searching tree for this problem is a binary tree whose vertices represent using c_{ij} or not. As long as the lower bound for possible tours using c_{ij} is less than the lower bound for the tours not using c_{ij}, we do not need to look at the subtree of possible tour not using c_{ij}.

If we use c_{wx} of C_2 to construct a tour, then the other entries of Row w and Column x cannot be used. So we delete Row w and Column x. The entry c_{xw} must be set to ∞ to avoid a cycle of length 2. The new smallest value of the reduced matrix in Row x is 1, and so we subtract 1 from Row x to obtain the cost matrix

$$C_{21} = \begin{array}{c|ccc} & w & y & z \\ \hline x & \infty & 2 & 0 \\ y & 0 & \infty & 0 \\ z & 5 & 0 & \infty \end{array} \qquad L.B. = 16 + 1 = 17$$

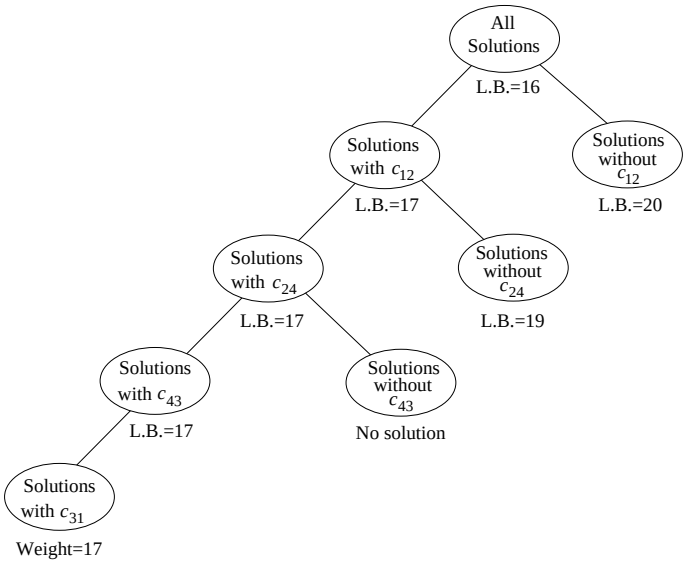
The lower bound of C_{21} is now 17. Since 17 is less than 20 (the lower bound of the tour does not contain (w, x)), we continue the tour building with (w, x) by choosing another entry. The new entry no need to connect with c_{wx} at this moment, that is, may not be the form c_{jw} or c_{xj}, but for simplicity we shall choose an entry in Row x. As before, we want to choose a 0 entry, say c_{xz}.

Again we have to choose whether c_{xz} is used or not. Not using c_{xz} will increase the lower bound by $2 + 0 = 2$ while using it will not increase the lower bound. Hence we construct the partial tour using c_{xz} (i.e., the arc (x, z)) along with (w, x). Same as before, we delete Row x as well as Column z and set $c_{zw} = \infty$ (to avoid the directed cycle $wxzw$). We obtain the new remaining cost matrix

$$C_{211} = \begin{array}{c|cc} & w & y \\ \hline y & 0 & \infty \\ z & \infty & 0 \end{array}$$

Now it is clear that we choose c_{zy} and c_{yw} to construct the tour $wxzyw$. This tour has a cost of 17, which attains the lower bound. Actually, we have no choice since not using either c_{zy} or c_{yw} forces us to use an ∞ entry. Finally, we obtain an optimal solution for the problem.

The following is our searching tree:



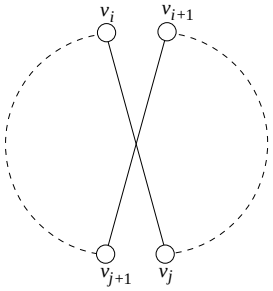
A general point should be made about how to optimize the branch and bound technique. At each stage of branching, we should choose the 0 entry whose removal will maximizes the increase in the lower bound. In C_2, not using c_{zy} will increase the lower bound by $3 + 3 = 6$. Hence c_{zy} would theoretically be a better entry than c_{wx} to use for the first branching, since the greater lower bound for the subtree of tours not using the edge zy makes it less likely that we need to check those possible tours in that subtree.

7.3.2 Near Optimal Solutions

Interchanging edges method: Firstly, choose an arbitrary Hamiltonian cycle $C = v_1 \cdots v_p v_1$. Then, search for another Hamiltonian cycle of smaller weight by suitably modifying C. For each i and j with $1 < i + 1 < j < p$, there is a new Hamiltonian cycle

$$C_{ij} = v_1 \cdots v_i v_j v_{j-1} \cdots v_{i+1} v_{j+1} \cdots v_p v_1$$

which obtained by deleting the edges $v_i v_{i+1}$ and $v_j v_{j+1}$ and adding the edges $v_i v_j$ and $v_{i+1} v_{j+1}$, as shown in the following figure.



If, for some i and j,

$$W(v_i v_j) + W(v_{i+1} v_{j+1}) < W(v_i v_{i+1}) + W(v_j v_{j+1})$$

the cycle C_{ij} will be an improvement of C.

Continue on this process until no more improvement is available, the resulting cycle is a near optimal solution. Also, this method can be repeated with different initial cycle.

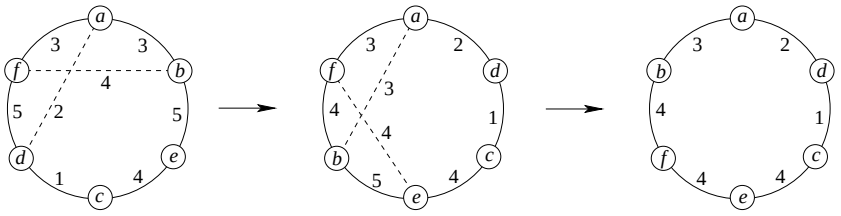
An indication of how good our solution is can be obtained by using spanning minimum tree. Suppose C is an optimal cycle in a weighted graph G. Then, for any vertex v, $C - v$ is a Hamiltonian path in $G - v$, and hence a spanning tree of $G - v$. Suppose T is a minimum spanning tree of $G - v$. If e_1 and e_2 are two edges incident with v such that $W(e_1) + W(e_2)$ is as small as possible, then $W(T) + W(e_1) + W(e_2)$ will be a lower bound on $W(C)$.

Example 7.3.2: Consider the weighted graph G with the cost matrix

$$\begin{pmatrix} \infty & 3 & 3 & 2 & 7 & 3 \\ 3 & \infty & 3 & 4 & 5 & 4 \\ 3 & 3 & \infty & 1 & 4 & 4 \\ 2 & 4 & 1 & \infty & 5 & 5 \\ 7 & 5 & 4 & 5 & \infty & 4 \\ 3 & 4 & 4 & 5 & 4 & \infty \end{pmatrix}$$

with respect to vertices a, b, c, d, e, f. Before applying the interchanging edges method to this graph, we find a lower bound on the optimal cycle in prior. By Prim's algorithm or Kruskal's algorithm, we find that the weight of the minimum spanning tree of $G - f$ is 10. Since af and ef are the edges incident with f such that $W(af) + W(ef) = 7$ attains the minimum value, hence the lower bound of the optimal cycle is 17.

Suppose we choose an initial Hamiltonian cycle $C = abecdfa$ of weight 21. Since $4 + 2 = W(bf) + W(ad) < W(ab) + W(df) = 3 + 5$, a new cycle $C' = adcebfa$ of weight 19 is obtained by interchanging edges. Moreover, $3 + 4 = W(ab) + W(ef) < W(af) + W(be) = 3 + 5$ lead to the construction of $C'' = adcefba$, which is of weight 18. We can see that it is very near to the lower bound. But we do not know whether it is an optimal cycle.



The second algorithm for finding a near optimal cycle of a weighted undirected complete graph K_p is called *quick construction.*

Quick Construction

Step 1: Pick any vertex u_1 as a starting cycle C_1 with one vertex and no edges. Choose the closest neighbor of u_1, say v_1. Let $C_2 = u_1v_1u_1$.

Step 2: Suppose a k-cycle C_k with $k \geq 2$ has been found. Find $v_k \notin V(C_k)$ that is closest to a vertex, say u_k, on C_k.

Step 3: Let C_{k+1} be the $(k+1)$-cycle obtained by inserting v_k immediately in front of u_k in C_k.

Step 4: Go to Step 2 until a Hamiltonian cycle is formed.

Example 7.3.3: Consider the graph in Example 7.3.2 again. We start with vertex a, which is C_1. Since d is closest to a, and so $C_2 = ada$. Also, c is closest to C_2 at d; thus $C_3 = acda$.

There are two vertices, namely b and f, both are three units from C_3. Suppose we choose b and insert it before c to obtain $C_4 = abcda$. Now f is still 3 units from a and we insert f before a obtaining $C_5 = abcdfa$. Finally, e is closest to c or f. Inserting e before f we obtain $C_6 = abcdefa$ with weight 19. (If we insert e before c to obtain $C_6 = abecdfa$, then the weight of C_6 is 21). It is an near optimal cycle.

Note that the weight of the tour generally depends on the starting vertex. One may apply the algorithm to different starting vertex and choose the shortest tour as a near optimal solution.

The following two theorems give bounds on the optimal cycle.

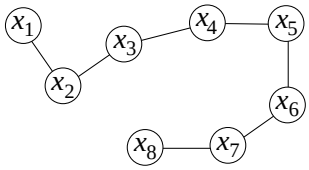
Theorem 7.3.4: *Suppose G is an undirected complete weighted graph satisfying triangle inequality. Then the weight of an optimal cycle in G is at most twice of the weight of the minimum spanning tree of G.*

Theorem 7.3.5: *Suppose G is an undirected complete weighted graph satisfying triangle inequality. Then the weight of the Hamiltonian cycle generated by the quick construction is less than twice the weight of the optimal Hamiltonian cycle.*

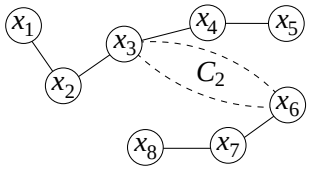
Example 7.3.6: Let $P = x_1x_2x_3x_4x_5x_6x_7x_8$. Suppose we choose $C_1 = x_6$ and $C_2 = x_6x_3x_6$. In this case, $v_1 = x_3$ and $S_1 = P$. The first edge on the path in S_1 from C_1 to x_3 is x_5x_6. Then $S_2 = S_1 - \{x_5x_6\} = \{x_1x_2x_3x_4x_5,\ x_6x_7x_8\}$. $W(C_1) = 0$ and $W(C_2) = 2W(x_3x_6) \leq 2W(x_5x_6)$ (because we choose x_3 not x_5).

Suppose $C_3 = x_6x_8x_3x_6$. Then $v_2 = x_8$ and $u_2 = x_3$. The first edge on the path in S_2 from C_2 to x_8 is x_6x_7. Then $S_3 = S_2 - \{x_6x_7\} = \{x_1x_2x_3x_4x_5,\ x_6,\ x_7x_8\}$. Since $W(x_3x_8) \leq W(x_6x_7)$ (because we choose x_3x_8 not x_6x_7), the increase of C_3 over C_2 is

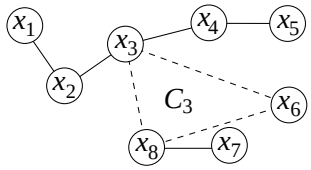
$$W(x_3x_8) + [W(x_8x_6) - W(x_3x_6)] \leq W(x_3x_8) + W(x_8x_3) \leq 2W(x_6x_7).$$



S_1 and $C_1 = x_6$



S_2 and $C_2 = x_6x_3x_6$



S_3 and $C_3 = x_6x_8x_3x_6$