“Graph theory”
Course for the master degree program “Geographic Information Systems”

Yulia Burkatovskaya

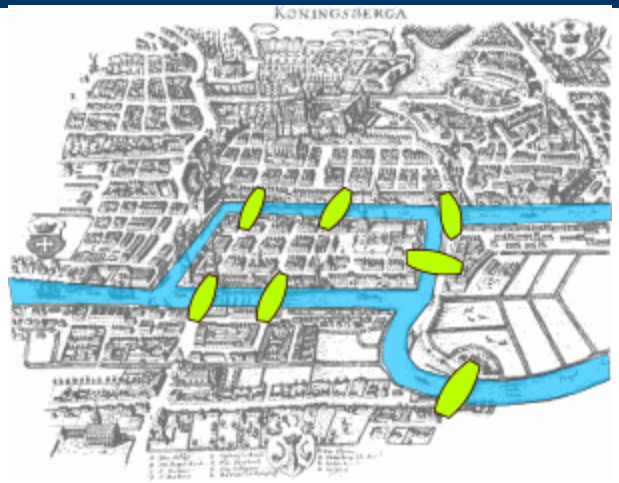
Department of Computer Engineering

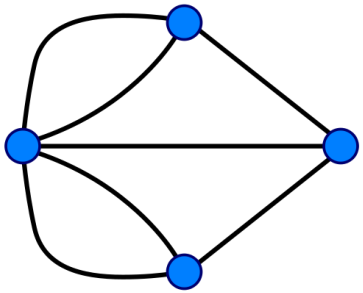
Associate professor

7. Euler graph

- Euler circuits
- Chinese Postman problem

Seven Bridges of Königsberg





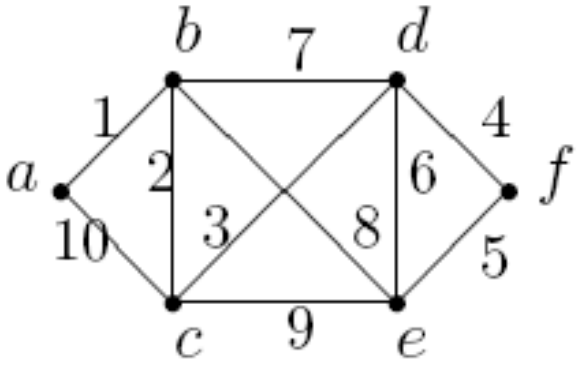
- The problem was to find a walk through the city that would cross each bridge once and only once.
- Its negative resolution by Leonhard Euler in 1735 laid the foundations of graph theory.
- The first problem of graph theory is connected with GIS!

7.1. Euler circuits

Euler circuits is any circuits in a graph which crosses each its edge exactly once.

Euler graph is a graph which has an Euler circuits.

Example. The numbers near edges show the order of their crossing.



Euler circuits

Theorem. An undirected graph is Euler if and only if all vertices have even degree.

Theorem. A directed graph is Euler if and only if for all vertices the in-degree is equal to the out-degree.

Fleury (1883).

Start. Choose any starting vertex *v*.

Step 1. If there are no edges incident to vertex *v* then go to the end.

Step 2. Choose the next edge (*v*,*u*) incident to vertex *v* which is not a bridge. If there are no such edges, choose the remaining edge (*v*,*u*) incident to vertex *v* .

Step 3. Move to vertex u and delete the chosen edge. Denote v=u. Go to step 1.

End. The sequence from which the edges were chosen forms an Euler circuit.

Hierholzer (1873)

Step 1. Choose any starting vertex *v*, and follow a trail of edges from that vertex until returning to *v*. The tour formed in this way may not cover all the vertices and edges of the initial graph.

Step 2. If there exists a vertex *u* that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from *u*, following unused edges until returning to *u*, and join the tour formed in this way to the previous tour. Repeat step 2 while it is possible.

Edmonds and Johnson (1973).

Step 1. Begin at any vertex s and construct a cycle C. This can be done by traversing any edge (s, x) incident to vertex s and marking this edge "used." Next traverse any unused edge incident to vertex x. Repeat this process of traversing unused edges until you return to vertex s.

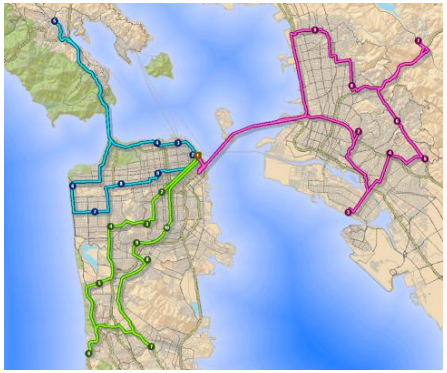
Step 2. If C contains all the edges of G, stop. If not, then the subgraph G' in which all edges of C are removed must be Euler since each vertex of C must have an even number of incident edges. Since G' is connected, there must be at least one vertex v in common with C.

Step 3. Starting at v, construct a cycle in G' say C'.

Step 4. Splice together the cycles C and C', calling the combined cycle C. Return to step 2.

7.2. The Chinese Postman problem

Before starting his or her route, a postal carrier must pick up the mail at the post office, then deliver the mail along each block on the route, and finally return to the post office. To make the job easier and more productive, every postal carrier would like to cover the route with as little walking as possible. Thus, the problem is to determine how to cover all the streets assigned and return to the starting point in the shortest total distance.



The Chinese Postman problem

Consider graph G(V,E) in which each edge represents a street on which mail must be delivered and each vertex represents an intersection.

Problem: finding a cycle in G which traverses each edge at least once in minimum total distance.

This problem was first discovered by a Chinese mathematician, Kwan Mei-Ko, and is popularly known as the Chinese postman problem (CPP).

7.2.1. The postman problem for undirected graphs

Consider graph $G(V,E)$.

Case 1. Graph G is even, i.e. all vertices have even degrees. A postman route is an Euler circuit in graph G.

Case 2. Graph G is not even. It can be transformed to even by repeating some edges.

Let $f(i, j) \geq 0$ denote the number of times that edge (i, j) is repeated by the postman. Edge (i, j) is traversed $f(i, j) + 1$ times by the postman.

Let $w(i, j)$ be the weight of edge (i, j).

The postman problem for undirected graphs

Construct a new graph G* = (V, E*) that contains $f(i, j) + 1$ copies of each edge (i, j) in graph G. An Euler tour of graph G* corresponds to a postman route in graph G.

The postman wishes to select values for the $f(i, j)$ variables so that:

a) the degrees of all vertices in G* are even;

b)

$$\sum_{(i,j)\in E} w(i,j) f(i,j) \rightarrow \min .$$

The postman problem for undirected graphs

- If vertex χ is an odd-degree vertex in graph G, an odd number of edges incident to vertex χ must be repeated by the postman, so that in graph G* vertex χ has even degree.
- If vertex χ is an even-degree vertex in graph G, an even number of edges (zero is an even number) incident to vertex χ must be repeated by the postman, so that in graph G* vertex χ has even degree.
- Graph G contains an even number of vertices with odd degree.
- Every path of repeated edges starts from an odd-degree vertex and ends at another odd-degree vertex.

The postman problem for undirected graphs

The postman must decide

- (a) which odd-degree vertices will be joined together by a path of repeated edges and
- (b) the precise composition of each such path.

The postman problem for undirected graphs

Theorem. A feasible solution to the postman problem is optimal if and only if

- (i) no more than one duplicate edge is added to any original edge and
- (ii) the length of the added edges in any cycle does not exceed one-half the length of the cycle.

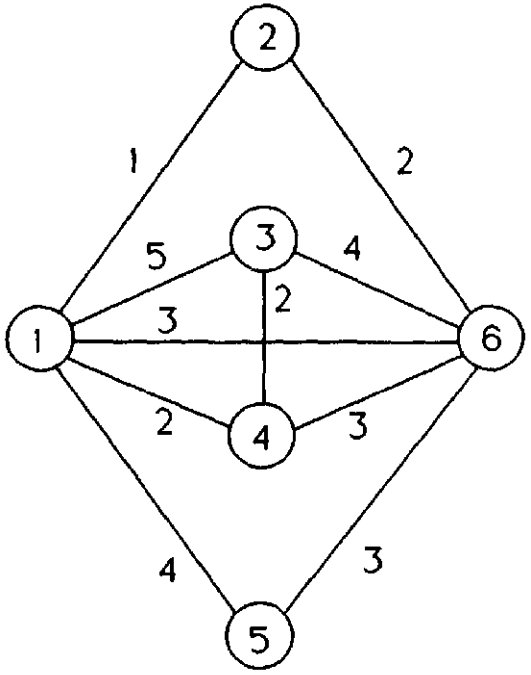
Lemma. If two feasible solutions satisfy (i) and (ii), then the lengths of their added edges are equal.

Lemma. Optimal solutions always exist.

Algorithm

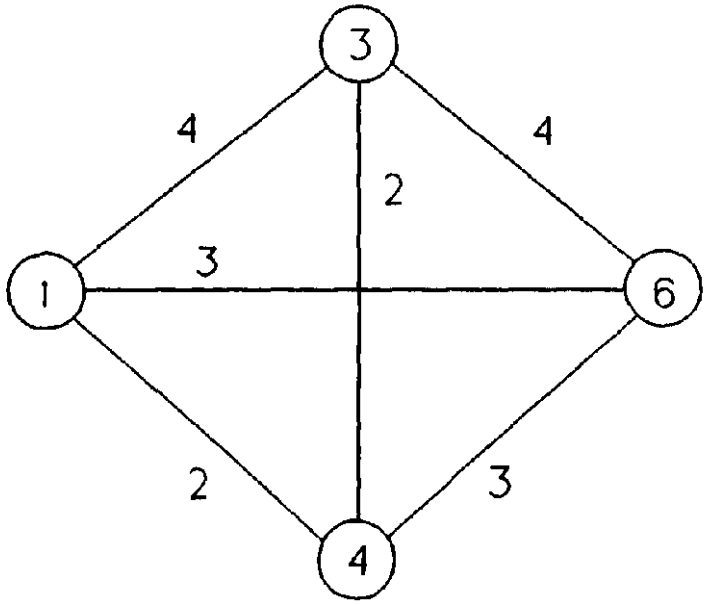
- Determine a shortest path between each pair of odd-degree vertices in graph G (Floyd algorithm).
- Construct a complete graph G' = (V, E') whose vertex set consists of all odd-degree vertices in G. Let the weight of each edge equal a very large number minus the length of a shortest path between the corresponding two vertices in graph G.
- Find a maximum-weight matching for graph G' using the maximum-weight matching algorithm.

Algorithm



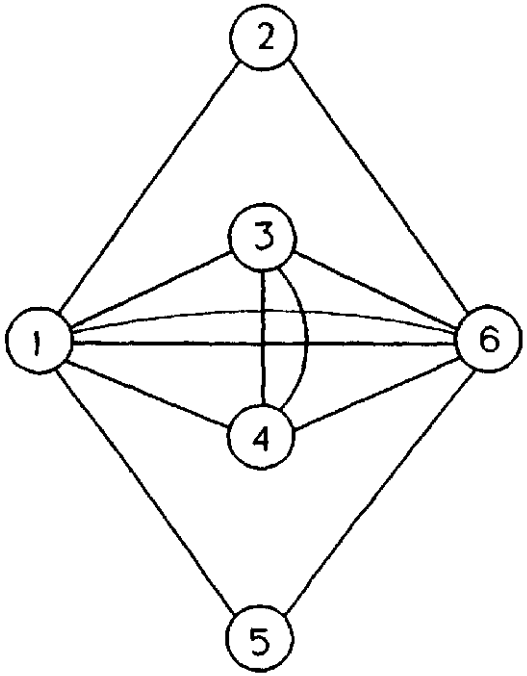
	1	2	3	4	5	6
1	0	1	4	2	4	3
2	1	0	5	3	5	2
3	4	5	0	2	7	4
4	2	3	2	0	6	3
5	4	5	7	6	0	3
6	3	2	4	3	3	0

Algorithm



Matching	Weight
(1, 3), (4, 6)	4 + 3 = 7
(1, 4), (3, 6)	2 + 4 = 6
(1, 6), (3, 4)	3 + 2 = 5

Algorithm



7.2.2. The postman problem for directed graphs

Consider graph $G(V,E)$.

Case 1. Graph G is symmetric, i.e. for all vertices $d^-(v)=d^+(v)$. A postman route is an Euler circuit in graph G.

Case 2. Graph G is not symmetric. It can be transformed to symmetric by repeating some edges.

Let $f(i, j)\geq 0$ denote the number of times that edge (i, j) is repeated by the postman. Edge (i, j) is traversed $f(i, j) + 1$ times by the postman.

Let $w(i, j)$ be the weight of edge (i, j).

The postman problem for directed graphs

The postman wishes to select values for the *f*(*i*, *j*) variables so that:

$$\sum_{(i,j)\in E} w(i,j)f(i,j) \to \min .$$

$$d^+(i) + \sum_{j\in V} f(i,j) = d^-(i) + \sum_{j\in V} f(j,i).$$

The postman problem for directed graphs

Rewrite the last condition:

$$\sum_{j \in V} (f(i,j) - f(j,i)) = d^+(i) - d^-(i) = D(i).$$

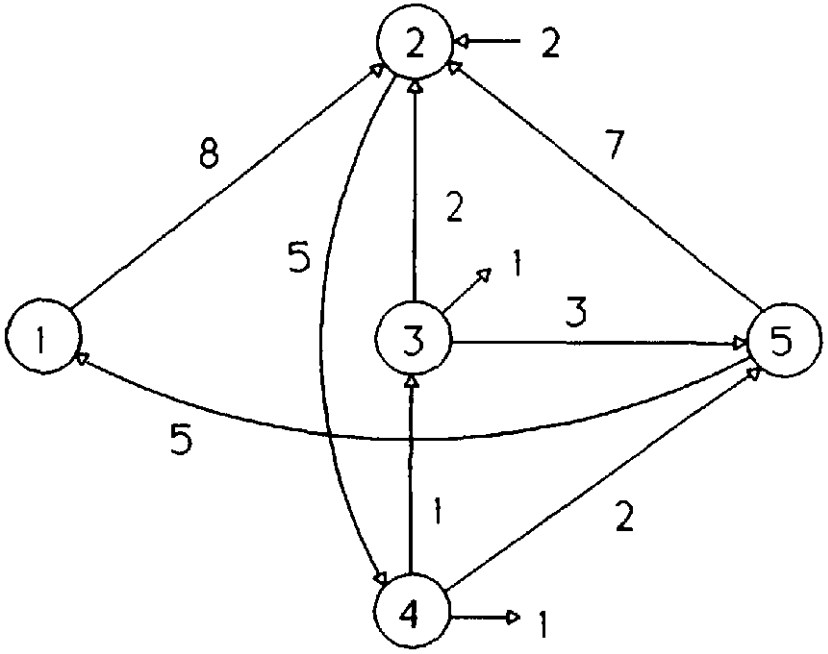
- D(*v*)<0: sinks with demand –D(*v*);
- D(*v*)>0: sources with supply D(*v*);
- D(*v*)=0: transshipment vertices.

All edge capacities are infinite. The task is to find a minimum-cost flow. It will produce optimal values for *f*(*i*, *j*).

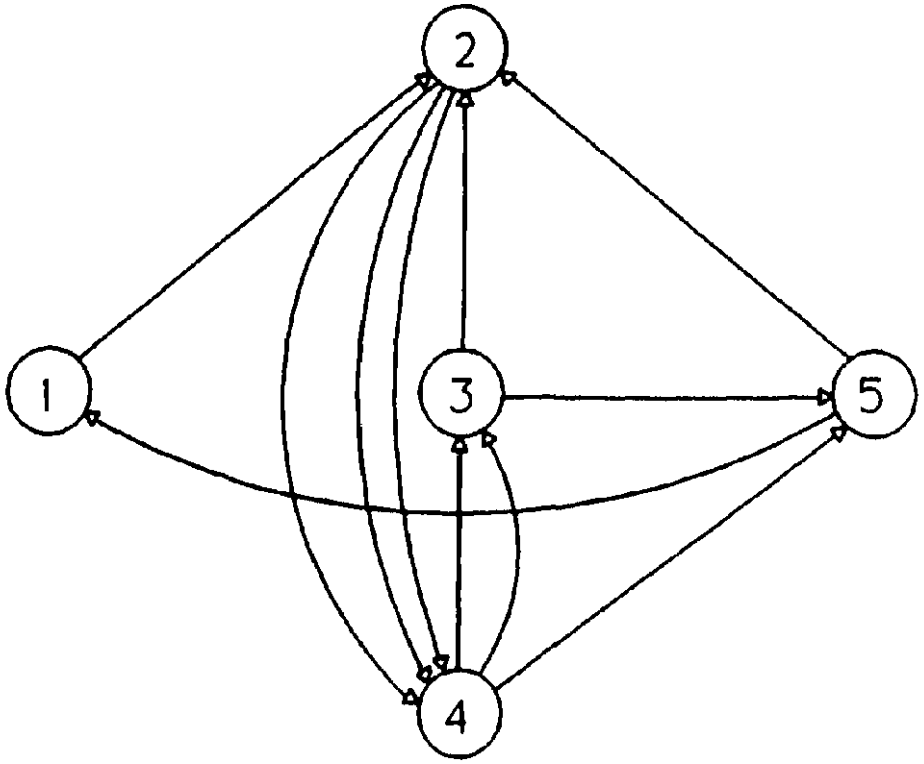
Algorithm

- Calculate D(*v*) for all vertices and find sinks, sources and transshipment vertices.
- Find any minimal-cost flow satisfying all the demands of sinks and all the supplies of sources. It gives the optimal integer values for the f(i, j) variables.
- Create a graph G* with f(i, j) + 1 copies of arc (i,j) for all (i,j). Graph G* is symmetric. An Euler tour of graph G* corresponds to an optimal postman route in graph G.

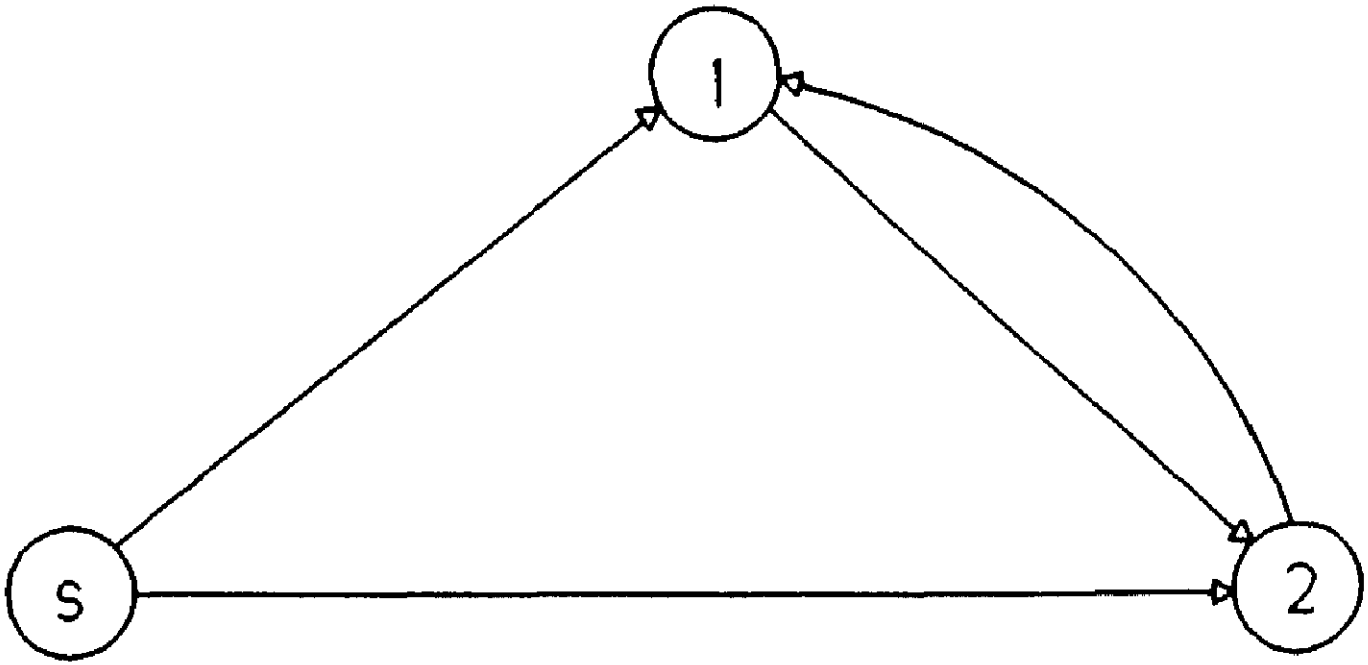
Algorithm



Algorithm



No postman route



7.2.3. The postman problem for mixed graphs

Consider graph $G(V,A,B)$, where A is the set of undirected edges, B is the set of directed edges.

Case 1. Graph *G* is even and symmetric. A postman route is a combination of Euler circuits in G(V,A) and G(V,B).

Case 2. Graph *G* is neither even no symmetric. The total enumeration or heuristic methods.

Case 3. Graph *G* is even but not symmetric. It can be transformed to symmetric digraph by directing and repeating some edges.

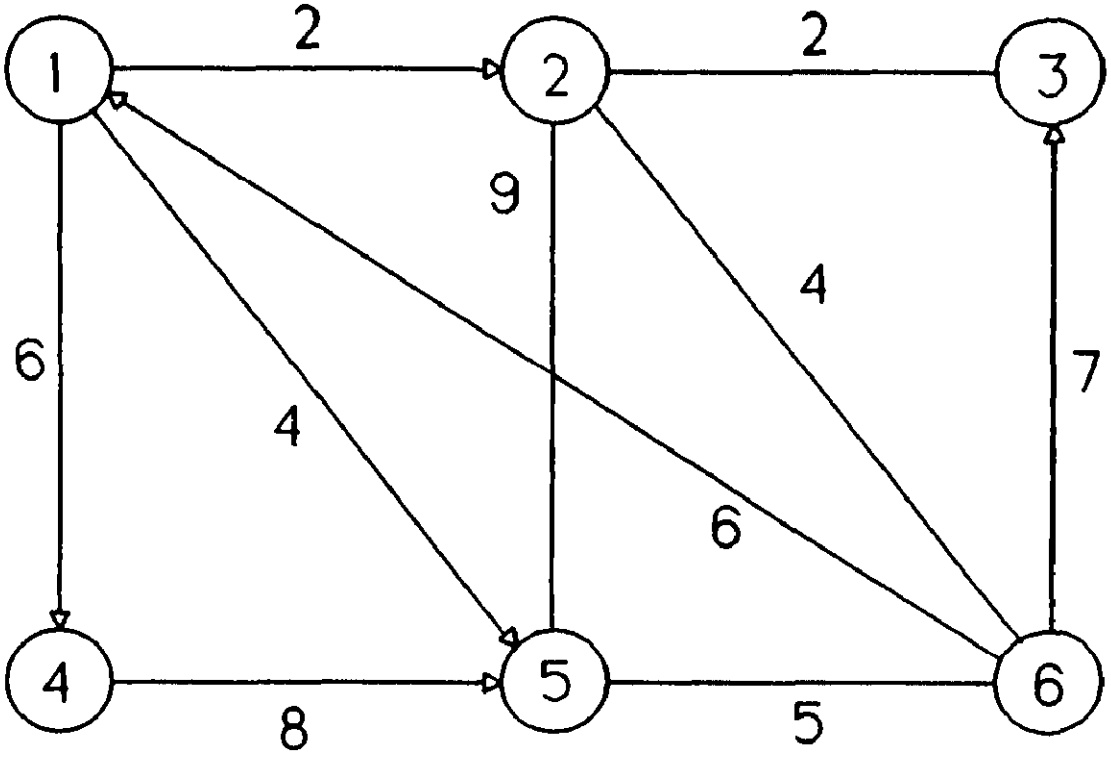
Algorithm

Select a direction for every edge from A and construct graph $G_D(V,E)$.

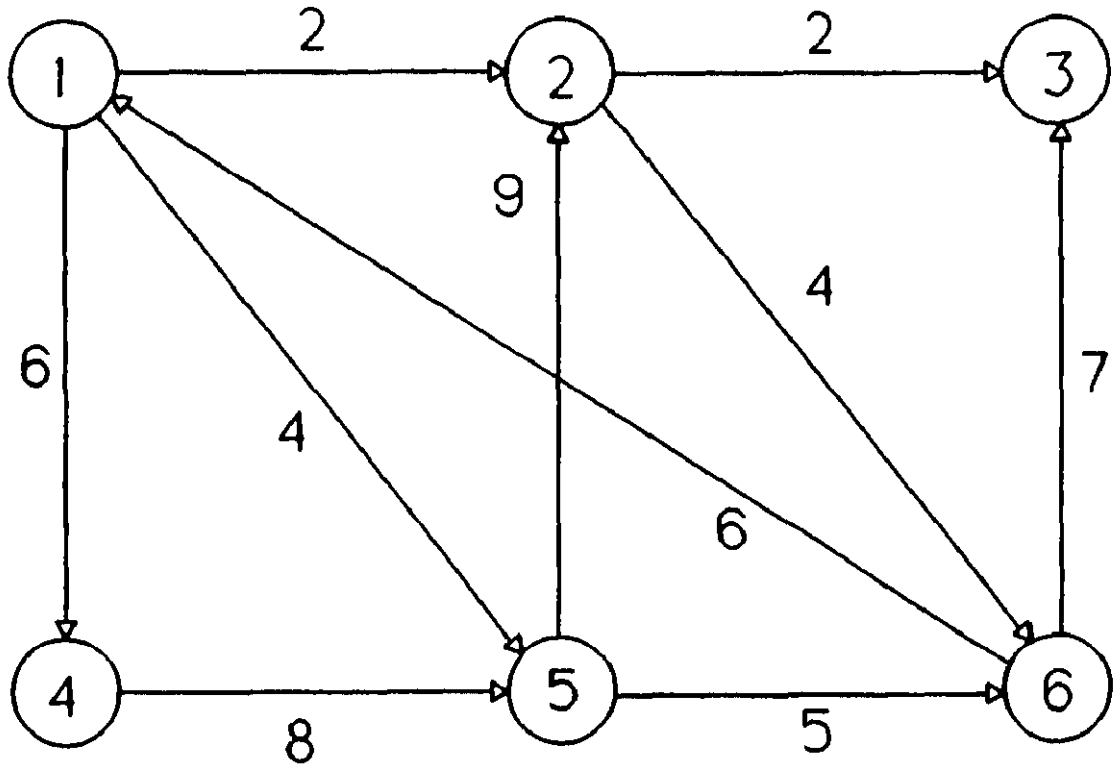
Case 1. $G_D(V,E)$ is symmetric. A postman route in graph G_D gives an Euler circuit in graph G.

Case 2. $G_D(V,E)$ is not symmetric.

Algorithm



Algorithm

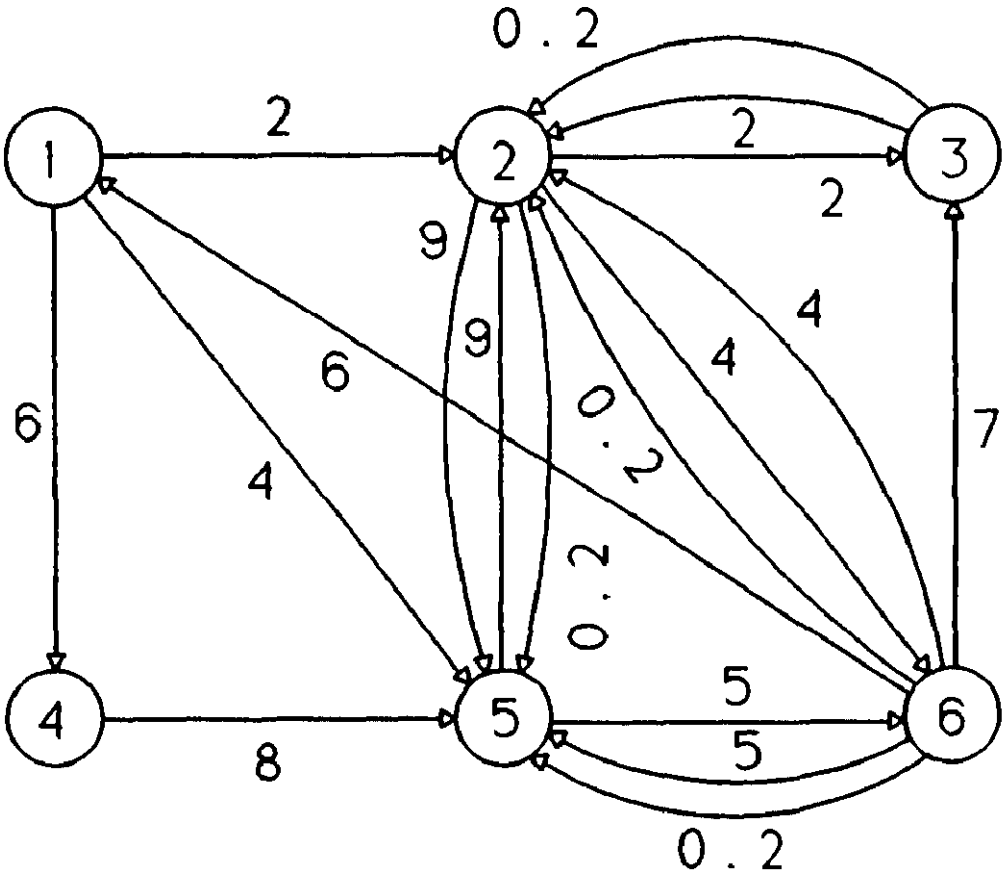


The postman problem for mixed graphs

Construct graph G’(V,E’) as follows.

- (a) For each arc (i, j) $\in$ B, place an edge (i,j) in B' with infinite capacity and cost equal to the length of (i,j).
- (b) For each arc (i, j) $\in$ A, create two directed arcs (i, j) and (j, i) in A'. Let each of these edges have infinite capacity and cost equal to the length of (i, j).
- (c) For each edge (i, j) $\in$ A, create a directed edge (j, i), in A' whose direction is the reverse of the direction assigned this edge in G_D. These edges are called *artificial edges*. Assign each artificial edge a zero cost and a capacity equal to two.

Algorithm



The postman problem for mixed graphs

Using the source supplies and sink demands defined above for graph G_D, find a minimum-cost flow in graph G' that satisfies all sink demands.

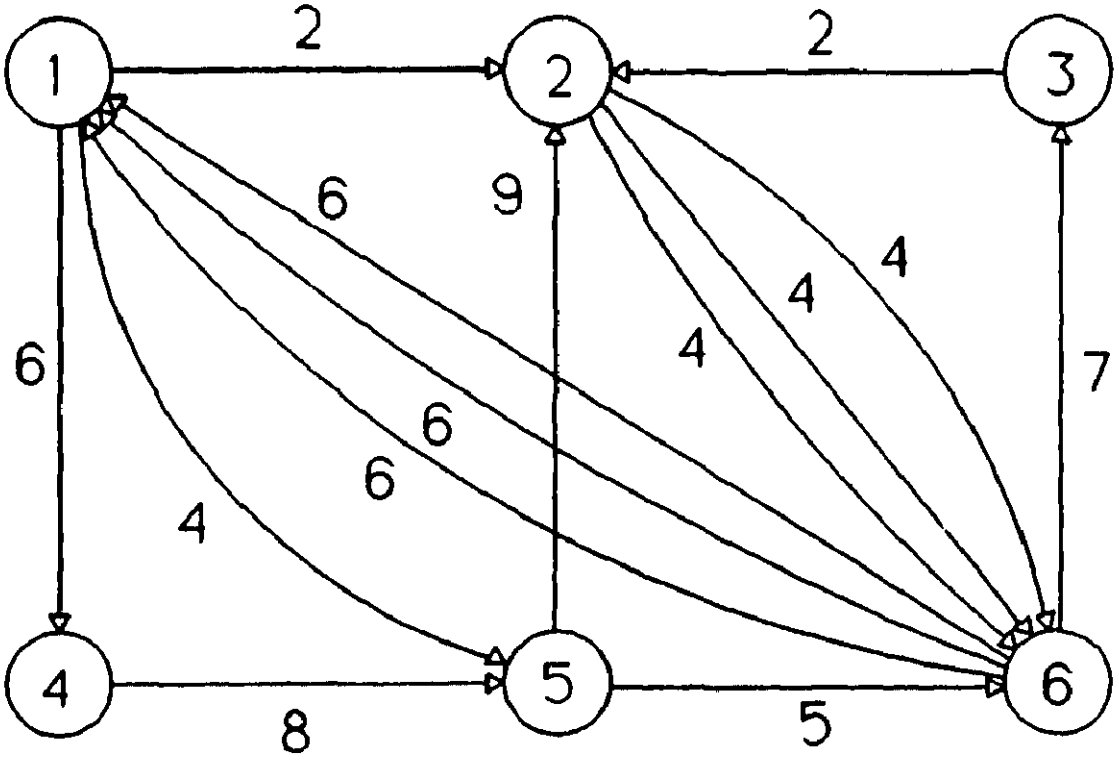
If no such flow exists, then no postman route exists. Otherwise, let f(i, j) denote the number of flow units sent through edge (i, j) in G’ in the minimum-cost flow produced by the minimum-cost flow algorithm.

The postman problem for mixed graphs

Create a graph G* as follows:

- (a) For each nonartificial edge (i, j) in G' place f(i, j) + 1 copies of edge (i, j) in graph G*.
- (b) If the flow in an artificial edge is two units, place one copy of this edge in graph G*.
- (c) If the flow in an artificial edge is zero, reverse the direction of this edge and place one copy of this edge in graph G*. (Thus, if no units traverse an artificial edge, the tentative direction assigned to this edge in G_D is retained; if two flow units traverse an artificial edge, the tentative direction assigned to this arc in G_D is reversed.)

Algorithm



The postman problem for mixed graphs

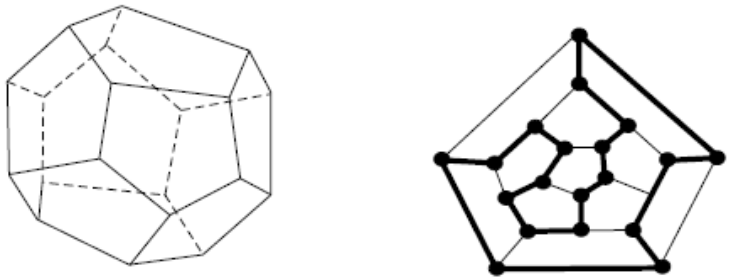
Graph G* is an even, symmetric, directed graph. An Euler tour of graph G* corresponds to an optimal postman route of the original graph G.

8. Hamiltonian graph

- Hamiltonian cycles
- Traveling Salesman problem

Icosian game

Hamiltonian graphs are named after Sir William Hamilton, an Irish Mathematician (1805−1865), who invented a puzzle, called the Icosian game, which he sold for 25 guineas to a game manufacturer in Dublin. The puzzle involved a dodecahedron on which each of the 20 vertices was labelled by the name of some capital city in the world. The aim of the game was to construct, using the edges of the dodecahedron a closed walk of all the cities which traversed each city exactly once, beginning and ending at the same city.



(a) (b)

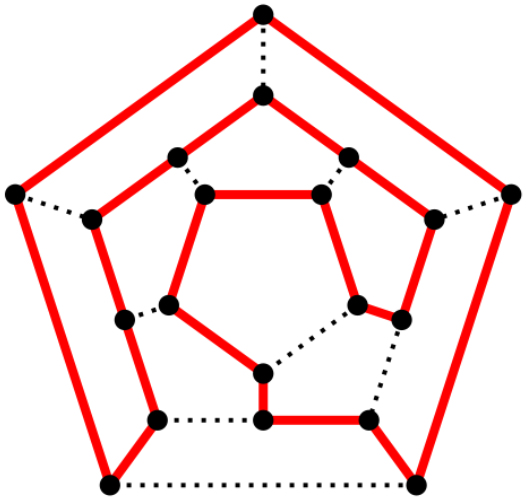
Dedecahedron and its graph shown with the Hamiltonian cycle

8.1. Hamiltonian cycle

Hamiltonian cycle is any cycle in a graph which visits each its vertex exactly once.

Hamiltonian graph is a graph which has a Hamiltonian cycle.

Example.



Theorems

- **Dirac** (1952)

  A simple graph with n vertices ($n \geq 3$) is Hamiltonian if every vertex has degree $n / 2$ or greater.

- **Ore** (1960)

  A graph with n vertices ($n \geq 3$) is Hamiltonian if, for every pair of non-adjacent vertices, the sum of their degrees is n or greater.

Theorems

- **Ghouila-Houiri** (1960)

  A strongly connected simple directed graph with n vertices is Hamiltonian if every vertex has a full degree greater than or equal to n.

- **Meyniel** (1973)

  A strongly connected simple directed graph with n vertices is Hamiltonian if the sum of full degrees of every pair of distinct non-adjacent vertices is greater than or equal to $2n - 1$.

Theorems

- **Ghouila-Houiri** (1960)

  A strongly connected simple directed graph with n vertices is Hamiltonian if every vertex has a full degree greater than or equal to n.

- **Meyniel** (1973)

  A strongly connected simple directed graph with n vertices is Hamiltonian if the sum of full degrees of every pair of distinct non-adjacent vertices is greater than or equal to $2n - 1$.

The enumeration method of Roberts and Flores

The method starts by forming a $k \times n$ matrix $M = [m_{ij}]$ where element m_{ij} is the i-th vertex (x_q say) for which an edge (x_j, x_q) exists in the graph $G(V,E)$. The vertices x in the set $\Gamma(x_j)$ can be arbitrarily arranged to form the entries of the j-th column of the M matrix. The number of rows k of the matrix M is then the largest out-degree of a vertex.

The enumeration method of Roberts and Flores

An initial vertex (say x_1) is chosen as the starting vertex and forms the first entry of the set *S* which will store the search path at any one time. The first vertex, (say vertex *a*) in column x_1 is added to *S*. Then the first feasible vertex (say vertex *b*) in column *a* is added to *S*, then the first feasible vertex (say vertex *c*) in column *b* is added to *S* and so on, where by "feasible" we mean a vertex which is not already in *S*.

The enumeration method of Roberts and Flores

Two possibilities now exist which will prevent any vertex being entered into $S = \{x_1,a,b,c,\ldots,x_{r-1},x_r\}$ at some stage r:

A) No vertex in column x_r is feasible,

B) The path represented by the sequence of vertices in S is of cardinality $n-1$, i.e. it forms a Hamiltonian path.

In case B) either:

(i) edge (x_r,x_j) exists in G and a Hamiltonian circuit is therefore found, or

(ii) edge (x_r,x_j) does not exist and no Hamiltonian circuit can be obtained.

In cases A and B(ii) backtracking must occur, whereas in case B(i) the search can either be stopped and the result printed.

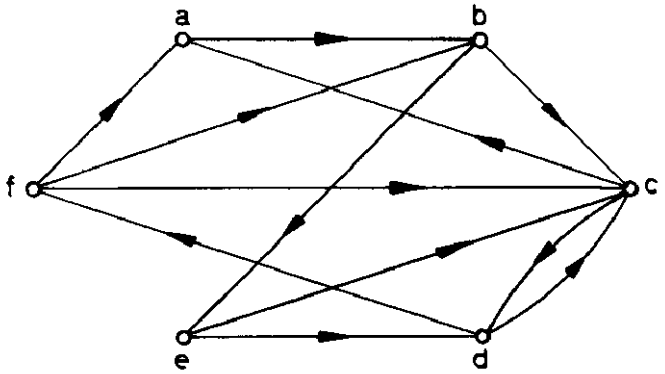
The enumeration method of Roberts and Flores

Backtracking involves the removal of the last-entered vertex x_r from S to produce the set $S = \{x_1,a,b,c,\ldots,x_{r-1}\}$ and the addition into S of the first feasible vertex following vertex x_r in column x_{r-1} of the M matrix. If no such feasible vertex exists a further backtracking step is taken and so on.

The end of the search occurs when the set S consists of the vertex x_1 only and no feasible vertex exists for adding into S so that a backtracking step would leave S empty. The Hamiltonian circuits found up to that time are then all the Hamiltonian circuits that exist in the graph.

Algorithm

	a	b	c	d	e	f
1	b	c	a	c	c	a
$\mathbf{M} = 2$	—	e	d	f	d	b
3	—	—	—	—	—	—

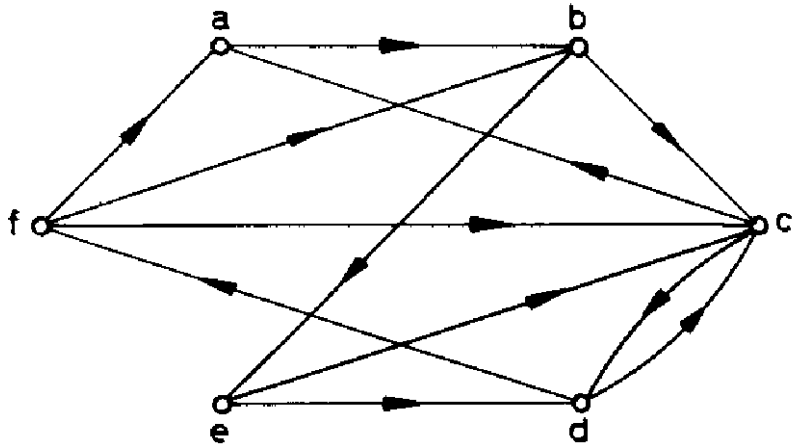


Algorithm

	Set S		*Notes*
1.	a	:	Add first feasible vertex in column a (i.e. vertex b)
2.	a, b	:	Add first feasible vertex in column b (i.e. vertex c)
3.	a, b, c	:	First vertex (a) in column c is infeasible ($a \in S$), add next vertex in column c (i.e. vertex d)
4.	a, b, c, d	:	Add vertex f
5.	a, b, c, d, f	:	No feasible vertex in column f exists. Backtrack
6.	a, b, c, d	:	No feasible vertex following vertex f in column d exists. Backtrack
7.	a, b, c	:	Similar to case above. Backtrack
8.	a, b	:	Add vertex e
9.	a, b, e	:	Add vertex c

Algorithm

10. a, b, e, c : Add vertex d
11. a, b, e, c, d : Add vertex f
12. $\underline{a, b, e, c, d, f}$: Hamiltonian path.

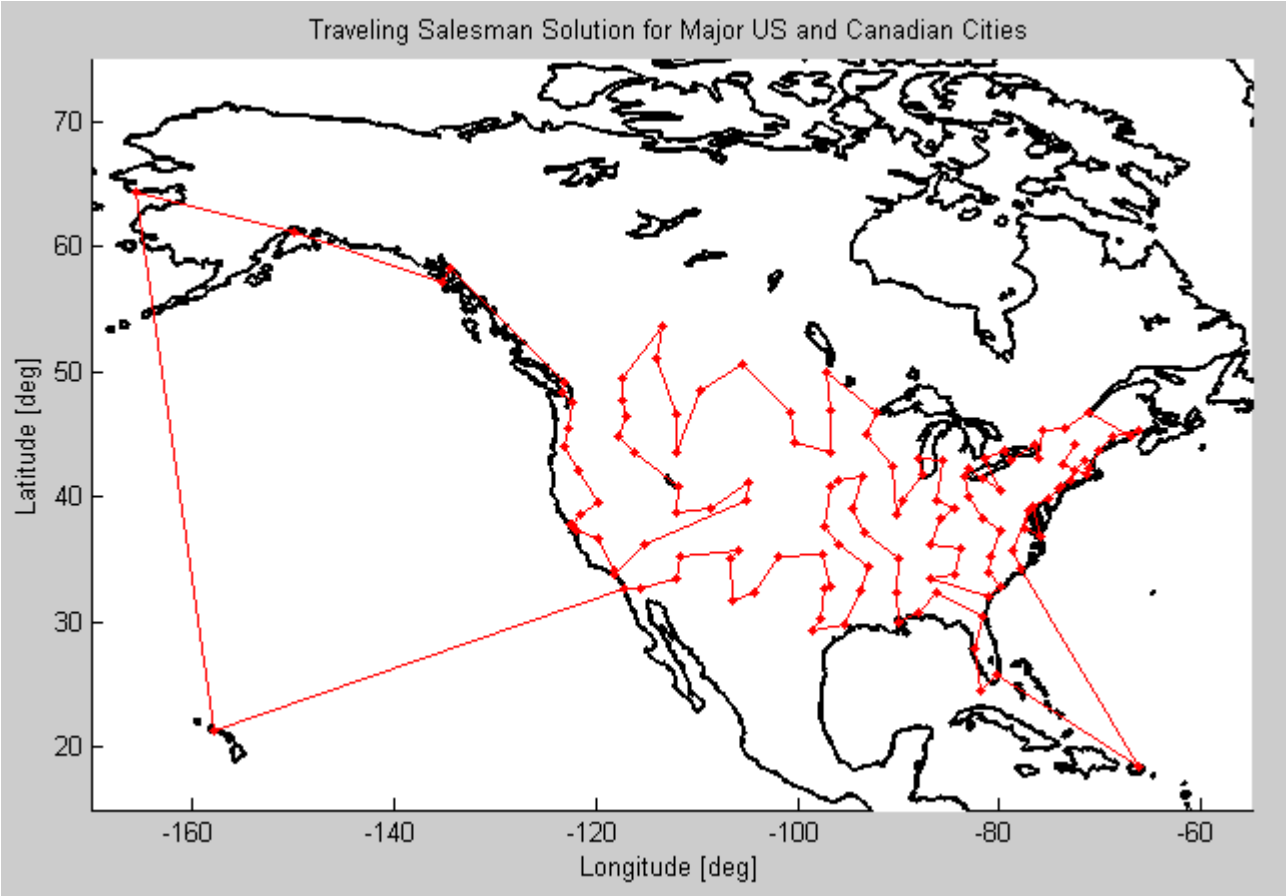


8.2. Traveling Salesman Problem

The **travelling salesman problem** (**TSP**) asks the following question:

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

Traveling Salesman Problem



Traveling Salesman Problem

Consider graph G(V,E) in which each vertex represents a city and each edge represents a road connecting two cities, and $d(x,y)$ is the weight of the edge (x,y) standing by the distance between cities x and y.

Problem: finding a cycle in G which visits each vertex once in minimum total distance.

This problem is **NP-hard**.

8.2.1. Some heuristic methods

- Cycle construction heuristics
- Cycle improvement heuristics

Nearest neighbor method

This is essentially a **greedy algorithm**.

Begin with any vertex *x* and find the vertex *y* so that *d*(*x*,*y*) is the smallest among all *y*.

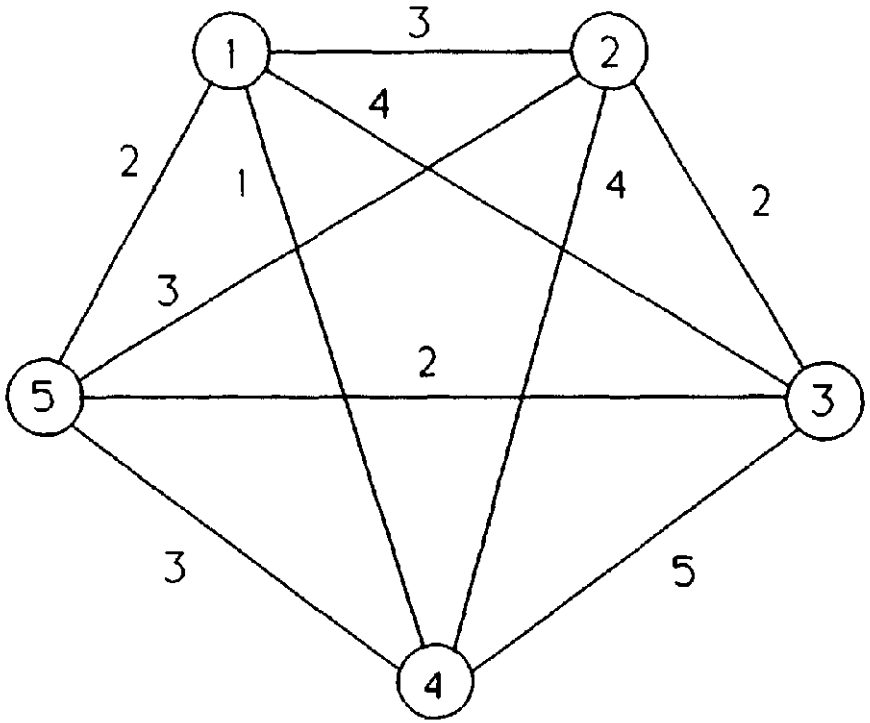
Next, find the closest vertex to y that is not already in the tour, say vertex *z*, and add edge (*y*, *z*) to the tour. Repeat this process until the last vertex is added and then join the first and last vertices by the unique edge between them.

Algorithm

Example.

The final cycle is

1–4–5–3–2–1.



Nearest insertion method

Select one vertex to start, say vertex i. Choose the nearest vertex, say j, and form the subtour $i - j - i$.

At each iteration, find the vertex κ not in the subtour that is closest to any vertex in the subtour. Find the edge (i,j) in the subtour which minimizes $d(i,k) + d(k,j) - d(i,j)$. Insert vertex κ between i and j.

Repeat this process until a tour is constructed. Note that in the iterative step, we try to add the least amount of distance to the current subtour by removing edge (i,j) and adding edges (i,k) and (k, j).

Algorithm

Example.

1–4–1

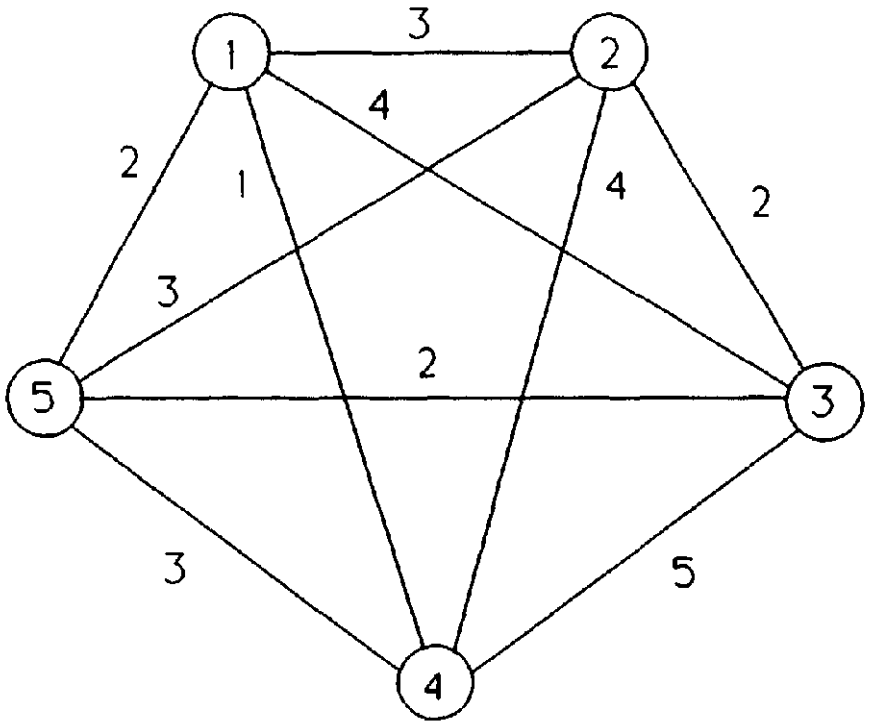
1–4–5–1

1–4–3–5–1

For the vertex 2

Arc	Incremental Cost
(1, 4)	3 + 4 − 1 = 6
(4, 3)	4 + 2 − 5 = 1
(3, 5)	3 + 2 − 2 = 3
(5, 1)	3 + 3 − 2 = 4

1–4–2–3–5–1

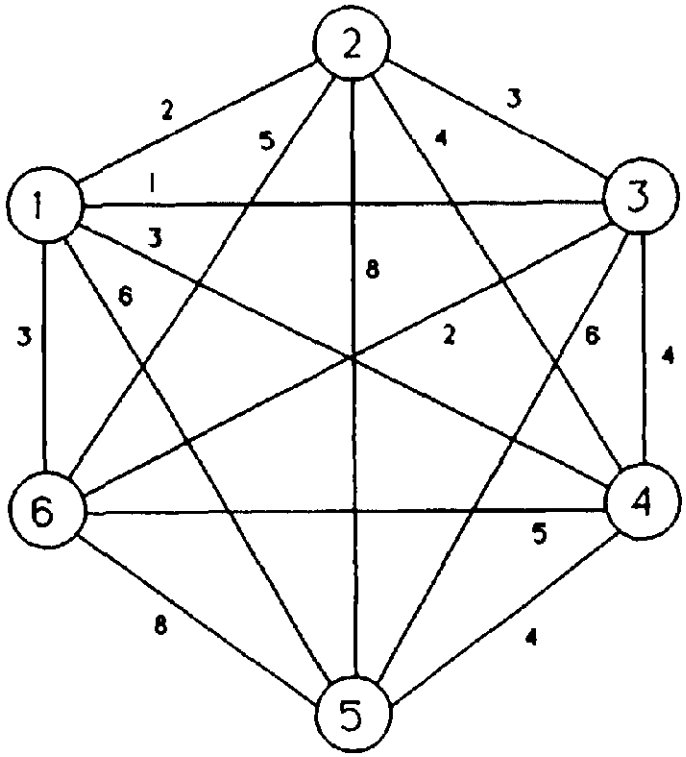


Christofides' Heuristic

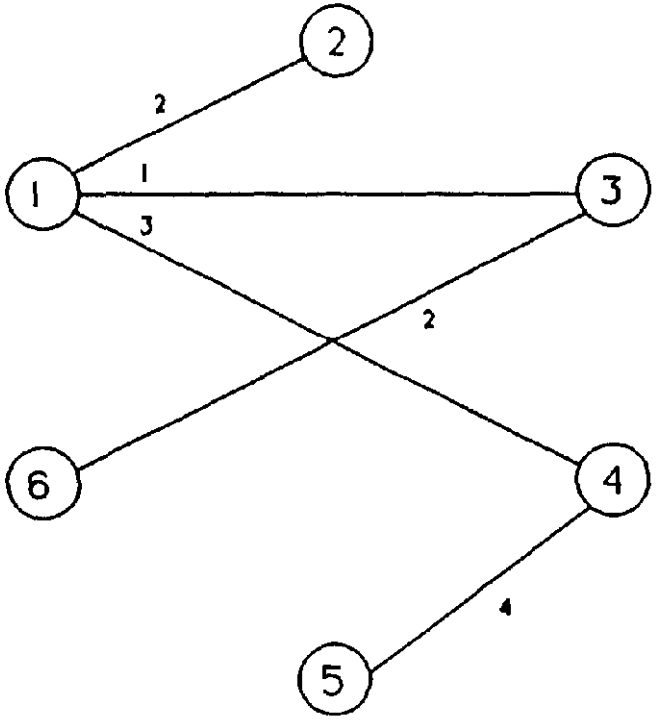
1. Construct the minimum spanning tree of the graph.
2. Find the minimum-cost matching of the odd-degree vertices in the spanning tree. Add the edges from the optimal matching to the tree to create an Euler graph.
3. Find an Euler circuits in this graph.
4. Transform the Euler circuits into a Hamiltonian cycle deleting repeated vertices.

Algorithm

Consider the TSP shown below.

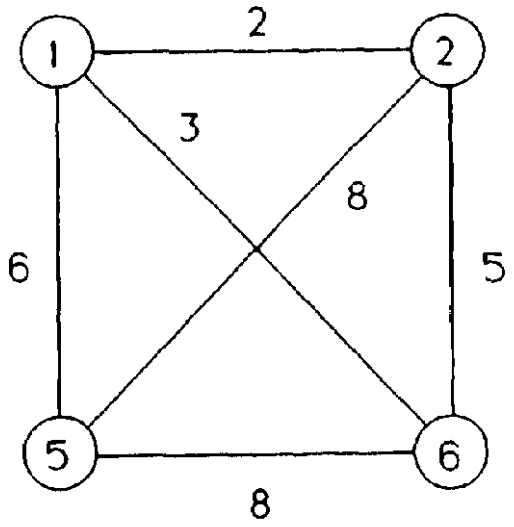


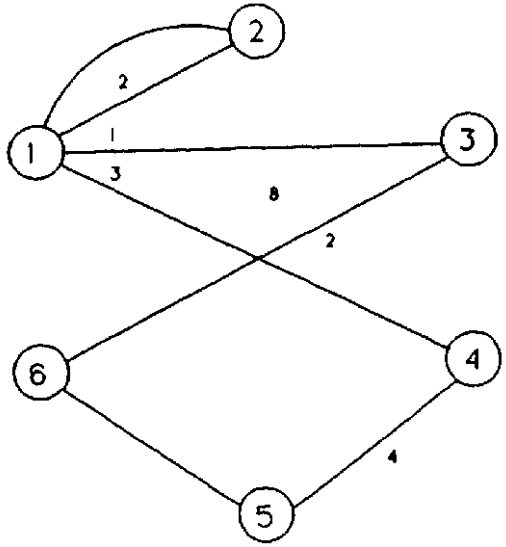
The minimal spanning tree is



Algorithm

The odd-degree vertices are 1, 2, 5, and 6. The matching problem on these vertices follows. The optimal matching is (1, 2) and (5, 6). Adding these edges to the minimal spanning tree creates the Euler graph





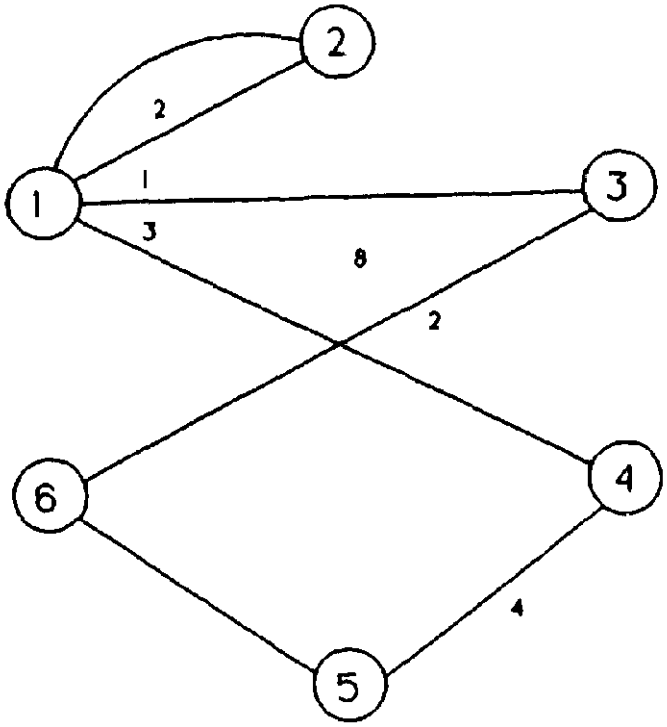
Algorithm

An Euler tour in this graph is

1-2-1-4-5-6-3-1.

When vertex 1 is repeated, we replace the path 2-1-4 by the edge (2, 4) to create the Hamiltonian cycle

1-2-4-5-6-3-1.



k-opt Heuristics

A *k*-change of a tour consists of deleting *k* edges and replacing them by *k* other edges to form a new tour.

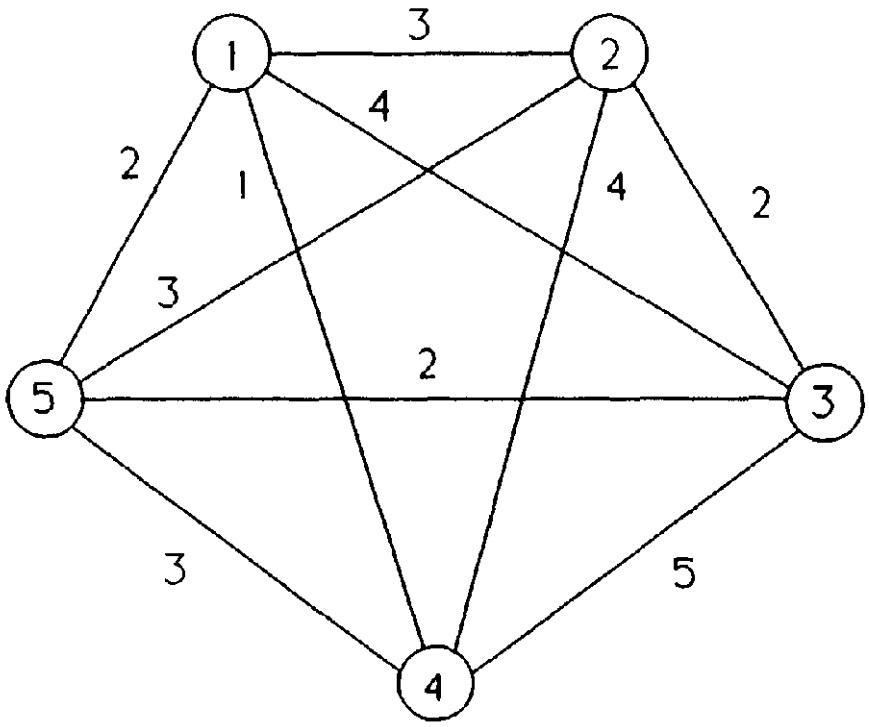
The heuristic procedure begins with any feasible tour. From this tour, all possible *k*-changes are examined, if a tour is found that has a lower cost than the current solution, it becomes the new solution. The process is repeated until no further *k*-change results in a better solution.

Algorithm

All 2-changes

(1-2-3-5-4-1 minimal)

Tour	Cost
1-3-2-4-5-1	15
1-4-3-2-5-1	13
1-2-4-3-5-1	16
1-2-5-4-3-1	18
1-2-3-5-4-1	11



8.2.2. Branch-and-bound method

- Low bound
- Branching
- Cutting branches

Low bound

Consider a graph G(V,E) with the matrix of weights W.

The minimum element of the row *i* gives us the minimum distance from *i* to another vertex. It can be understood as the minimum price we pay to leave the city *i*.

Example.

The minimum element are written on the right.

	a	b	c	d	e	f	
a		8	4	6	12	3	3
b	7		13	9	1	7	1
c	2	13		16	11	6	2
d	5	7	18		3	14	3
e	10	2	12	4		13	2
f	2	8	3	11	10		2

Low bound

Then we subtract the minimum elements from the elements of the corresponding rows.

The minimum element of the column *j* gives us the minimum distance from another vertex to *j*. It can be understood as the minimum price we pay to come to the city *j*.

Example.

The minimum element are written underneath.

	a	b	c	d	e	f
a		5	1	3	9	0
b	6		12	8	0	6
c	0	11		14	9	4
d	3	5	16		0	11
e	8	0	10	2		11
f	0	6	1	9	8	
	0	0	1	2	0	0

Low bound

Then we subtract the minimum elements from the elements of the corresponding columns. This process is called **reduction** of the matrix. The obtained matrix is called **reduced matrix**.

Example.

$$A = \begin{array}{c|cccccc} & a & b & c & d & e & f \\ \hline a & & 5 & 0 & 1 & 9 & 0 \\ b & 6 & & 11 & 6 & 0 & 6 \\ c & 0 & 11 & & 12 & 9 & 4 \\ d & 3 & 5 & 15 & & 0 & 11 \\ e & 8 & 0 & 9 & 0 & & 11 \\ f & 0 & 6 & 0 & 7 & 8 & \end{array}$$

Low bound

The low bound of the TSP solution is the sum of the minimum elements found during reducing of the weight matrix.

Example.

$$\underline{C}(A) = 3 + 1 + 2 + 3 + 2 + 2 + 0 + 0 + 1 + 2 + 0 + 0 = 16.$$

Branching

A problem M is divided into two subproblems:

1) we include the edge (i,j) to the tour; then we delete the row i and the column j from the matrix M. The obtained matrix M' is reduced and the sum of the minimum elements of its rows and columns is added to the low bound of the problem M, the result is the low bound of the problem M';
2) we don't include the edge (i,j) to the tour; then we replace the element (i,j) of the matrix M by infinity. The obtained matrix M'' is reduced: the minimum elements of the row i and of the column j are subtracted from the row and the column respectively and added to the low bound of the problem M, the result is the low bound of the problem M".

Branching

How to choose the edge of the branching?

1) M(i,j)=0.
2) The edge (i,j) cannot close a cycle with other edges included in the tour except of the last edge.
3) The **index** of the edge (i,j) is the sum of the minimum elements of the row i and the column j except of M(i,j):

$$\delta(i,j) = \min_{k \neq j} M(i,k) + \min_{k \neq i} M(k,j).$$

4) We choose the edge with the maximun index to obtain the maximum value of the low bound of the subproblem M". If the edge closes a cycle with other edges included in the tour then we replace it by infinity and reduce the matrix.

Branching

Example. Calculate the indexes of the elements of the matrix M. The edge (b,e) has the maximun index.

$A =$

	a	b	c	d	e	f
a		5	0(0)	1	9	0(4)
b	6		11	6	0(6)	6
c	0(4)	11		12	9	4
d	3	5	15		0(3)	11
e	8	0(5)	9	0(1)		11
f	0(0)	6	0(0)	7	8	

$$\delta(a,c) = 0 + 0 = 0;$$
$$\delta(a,f) = 0 + 4 = 4;$$
$$\delta(b,e) = 6 + 0 = 6;$$
$$\delta(c,a) = 4 + 0 = 4;$$
$$\delta(d,e) = 3 + 0 = 3;$$
$$\delta(e,b) = 0 + 5 = 5;$$
$$\delta(e,d) = 0 + 1 = 1;$$
$$\delta(f,a) = 0 + 0 = 0;$$
$$\delta(f,c) = 0 + 0 = 0.$$

Branching

Example. Include the edge (b,e). into the tour.

$B =$

	a	b	c	d	f	
a		5	0	1	0	0
c	0	11		12	4	0
d	3	5	15		11	3
e	8	0	9	0	11	0
f	0	6	0	7		0

	a	b	c	d	f
a		5	0	1	0
c	0	11		12	4
d	0	2	12		8
e	8	0	9	0	11
f	0	6	0	7	

$$\underline{C}(B) = \underline{C}(A) + 3 = 16 + 3 = 19.$$

Branching

Example. Don't include (b,e) into the tour.

$C =$

	a	b	c	d	e	f
a		5	0	1	9	0
b	6		11	6		6
c	0	11		12	9	4
d	3	5	15		0	11
e	8	0	9	0		11
f	0	6	0	7	8	

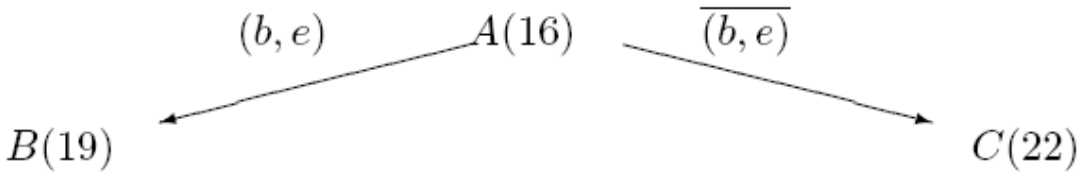
$C =$

	a	b	c	d	e	f
a		5	0	1	9	0
b	0		5	0		0
c	0	11		12	9	4
d	3	5	15		0	11
e	8	0	9	0		11
f	0	6	0	7	8	

$$\underline{C}(C) = \underline{C}(A) + 6 = 16 + 6 = 22.$$

Branching

Example. The decision tree.



Cutting branches

To cut brunches we need the upper bound of the TSP solution. For example, it can be the distance of any tour. A branch is cut if its lower bound is not less then the upper bound.

Example.

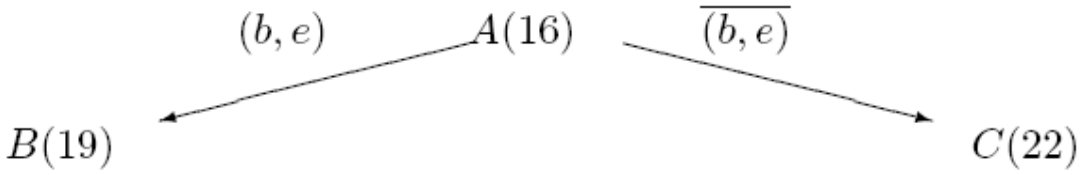
	a	b	c	d	e	f
a		8	4	6	12	3
b	7		13	9	1	7
c	2	13		16	11	6
d	5	7	18		3	14
e	10	2	12	4		13
f	2	8	3	11	10	

$$abcdefa$$

$$\overline{C} = 8 + 13 + 16 + 3 + 13 + 2 = 55.$$

Cutting branches

Example. Subproblems B and C have the lower bounds les then 55 so we don’t cut the branches.



Cutting branches

If a new tour is obtained and its lower bound is less then the upper bound then we change the upper bound.

Example.