

Eulerian and Hamiltonian Paths

1. Euler paths and circuits

1.1. The Königsberg Bridge Problem

Königsberg was a town in Prussia, divided in four land regions by the river Pregel. The regions were connected with seven bridges as shown in figure 1(a). The problem is to find a tour through the town that crosses each bridge exactly once. Leonhard Euler gave a formal solution for the problem and –as it is believed- established the graph theory field in mathematics.

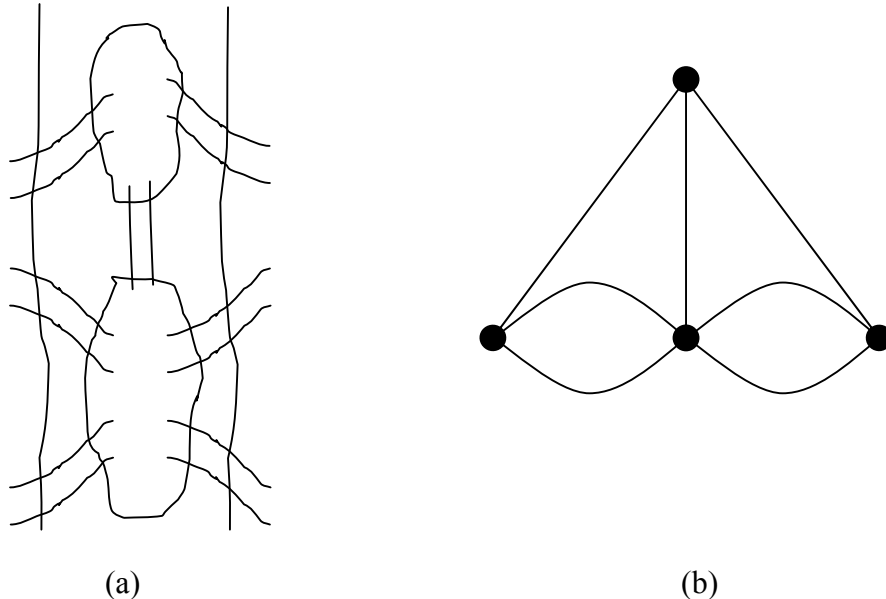


Figure 1: The bridges of Königsberg (a) and corresponding graph (b)

1.2. Defining Euler paths

Obviously, the problem is equivalent with that of finding a path in the graph of figure 1(b) such that it crosses each edge exactly one time. Instead of an exhaustive search of every path, Euler found out a very simple criterion for checking the existence of such paths in a graph. As a result, paths with this property took his name.

Definition 1: An Euler path is a path that crosses each edge of the graph exactly once. If the path is closed, we have an Euler circuit.

In order to proceed to Euler's theorem for checking the existence of Euler paths, we define the notion of a vertex's degree.

Definition 2: The degree of a vertex u in a graph equals to the number of edges attached to vertex u . A loop contributes 2 to its vertex's degree.

1.3. Checking the existence of an Euler path

The existence of an Euler path in a graph is directly related to the degrees of the graph's vertices. Euler formulated the three following theorems of which the first two set a sufficient and necessary condition for the existence of an Euler circuit or path in a graph respectively.

Theorem 1: An undirected graph has at least one Euler path iff it is connected and has two or zero vertices of odd degree.

Theorem 2: An undirected graph has an Euler circuit iff it is connected and has zero vertices of odd degree.

Theorem 3: The sum of the degrees of every vertex of a graph is even and equals to twice the number of edges.

A proof for Theorems 1.1 and 1.2(as a special case of 1.1) follows.

Proof:

⊢ An Euler trail exists. As the path is traversed, each time that a vertex is reached we cross two edges attached to the vertex and have not been crossed yet. Thus, all vertices, except maybe the starting vertex a and the ending vertex b , have even degrees. If $a = b$ we have an Euler circuit and if $a \neq b$ we have an open path.

⊔ Induction on the number of vertices

Induction beginning: $|V| = 2$. Elementary case

Induction assumption: For $|V| \leq n$ the theorem holds.

Induction step: $|V| = n + 1$. If we have two vertices with odd degrees a and b we can find a longest path beginning at a and ending at b . For any left edges, we can use the condition assumed by the induction. If we have no vertices of odd degree we use the same method except a is the beginning and ending vertex.

Example: Figure 2 shows some graphs indicating the distinct cases examined by the preceding theorems. Graph (a) has an Euler circuit, graph (b) has an Euler path but not an Euler circuit and graph (c) has neither a circuit nor a path.

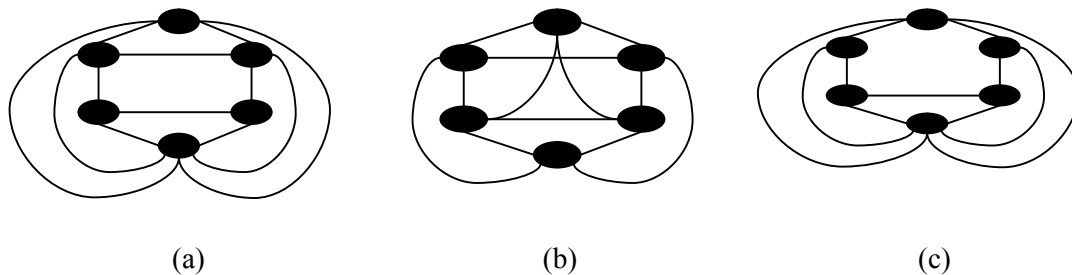


Figure 2: A graph containing an Euler circuit (a), one containing an Euler path (b) and a non-Eulerian graph (c)

1.4. Finding an Euler path

There are several ways to find an Euler path in a given graph. Since it is a relatively simple problem it can solve intuitively respecting a few guidelines:

- Always leave one edge available to get back to the starting vertex (for circuits) or to the other odd vertex (for paths) as the last step.
- Don't use an edge to go to a vertex unless there is another edge available to leave that vertex (except for the last step).

This technique certainly gives results for simple graphs but we surely need a formal algorithm to find Euler circuits. In the following sections we present two algorithms serving that purpose.

Fluery's algorithm

Fluery's algorithm can be applied in searches for both Euler circuits and paths.

FLUERY

Input: A connected graph $G = (V, E)$ with no vertices of odd degree

Output: A sequence P of vertices and their connecting edges indicating the Euler circuit.

1 Choose any vertex u_0 in G .

2 $P = u_0$

3 if $P_i = u_0 e_1 u_1 e_2 \dots e_i u_i$ choose edge e_{i+1} so that

1. e_{i+1} is adjacent to e_i
2. Removal of e_{i+1} does not disconnect $G - \{e_1, \dots, e_i\}$ (i.e. e_{i+1} is not a bridge).

4 if P includes all edges in E return P

5 goto 3

The algorithm for finding an Euler path instead of a circuit is almost identical to the one just described. Their only difference lies in step 1 where we must choose one of the vertices of odd degree as the beginning vertex.

Example: Figure 3 demonstrates some important steps in the process described by the algorithm 1.4. The numbers that label each edge indicate their index in the Euler circuit.

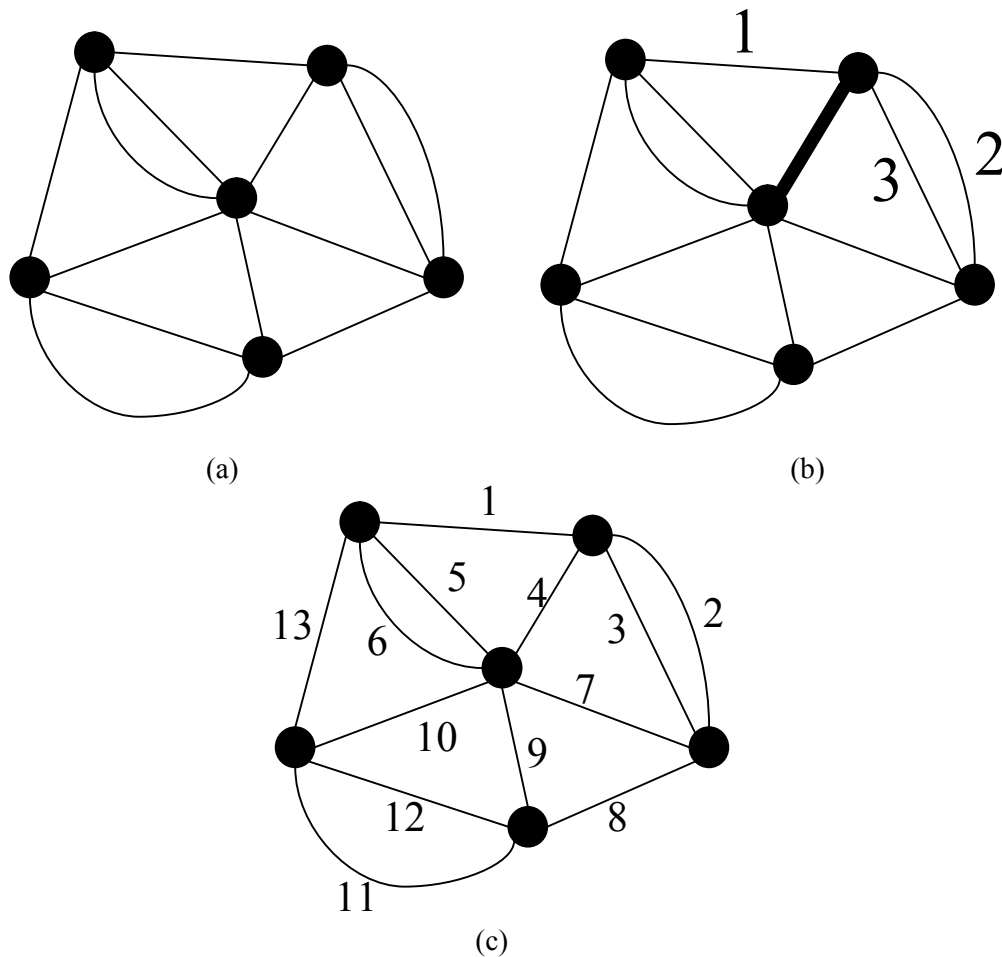


Figure 3: The input graph (a). The bold line is a bridge but we have no other choice (b). The output graph, showing the sequence of edges in the Euler circuit (c).

A constructive algorithm

The ideas used in the proof of Euler's theorem can lead us to a recursive constructive algorithm to find an Euler path in an Eulerian graph.

CONSTRUCT

Input: A connected graph $G = (V, E)$ with two vertices of odd degree.

Output: The graph with its edges labeled according to their order of appearance in the path found.

- 1 Find a simple cycle in G .
- 2 Delete the edges belonging in C .
- 3 Apply algorithm to the remaining graph.
- 4 Amalgamate Euler cycles found to obtain the complete Euler cycle.

The process described can be extended to handle Euler paths as well. We are obliged to start from a vertex of odd degree u . Then we add a “phantom edge” between the vertices of odd degree and thus ensure that every vertex has an even degree. We find the circuit starting and ending at u and then remove the phantom edge to obtain the Euler path.

Example: Figure 4 demonstrates the constructive algorithm’s steps in a graph. 4(a) shows the initial graph, and 4(b), 4(c) show the simple cycle found. 4(d) shows the next cycle and 4(e) the amalgamation of the two cycles found. The process continues and we result to the graph of figure 4(i).

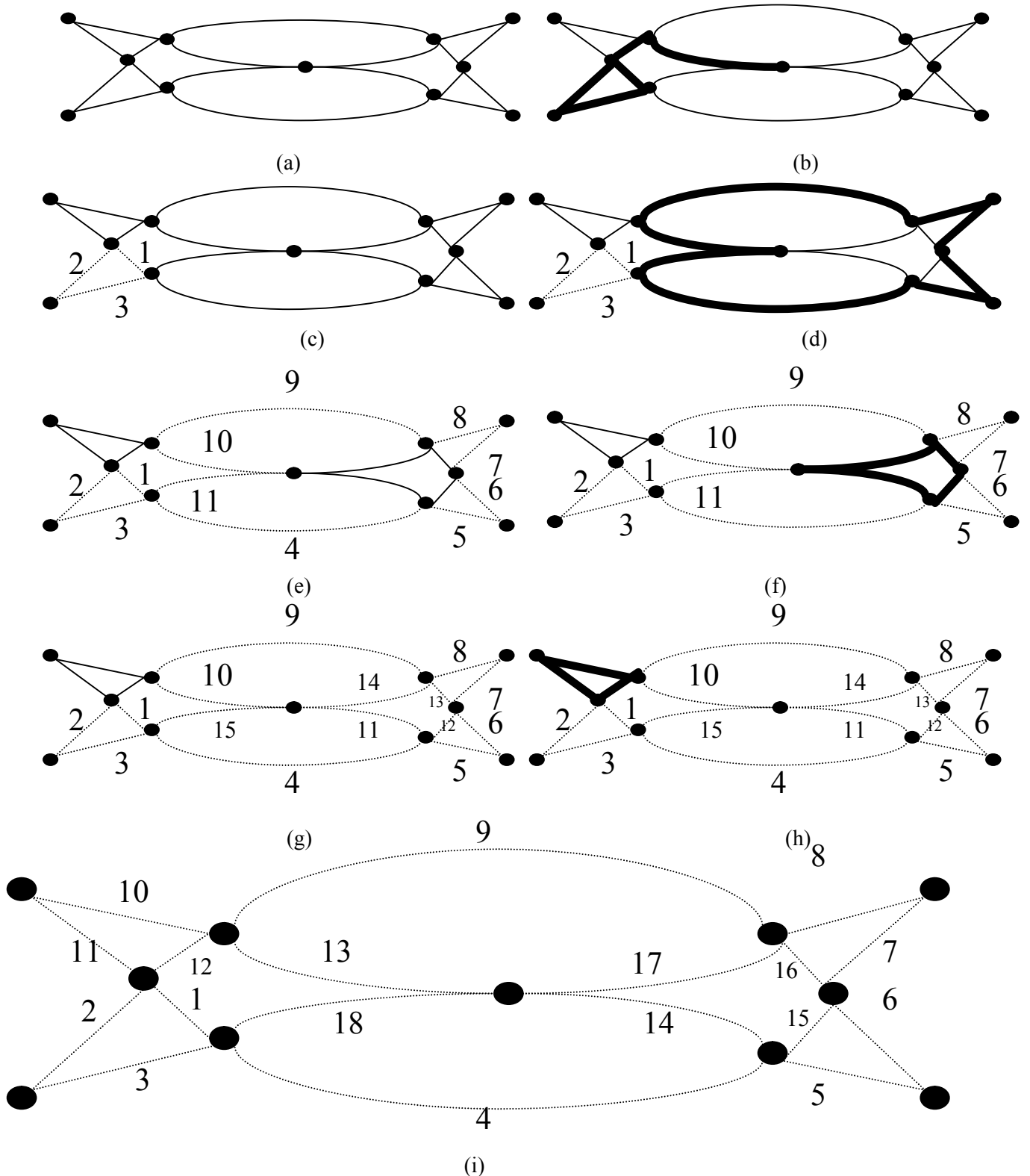


Figure 4: Applying the algorithm to graph 4(a) and resulting to graph 4(i)

1.5 Expanding to directed graphs

The results aforementioned can easily be adapted to directed graphs. We proceed to two definitions analogous to definition 1.1.

Definition 3: The in-degree of a vertex in a directed graph is the number of edges incoming to that vertex.

Definition 4: The out-degree of a vertex in a directed graph is the number of edges outgoing from that vertex.

The condition that a directed graph must satisfy to have an Euler circuit is defined by the following theorem.

Theorem 4: A directed graph G has an Euler circuit iff it is connected and for every vertex u in G $\text{in-degree}(u) = \text{out-degree}(u)$.

Example: An interesting problem (and with some practical worth as well) is the following: Place 2^n binary digits in a circular table in such a way that the 2^n sequences of n digits are all different. The solution is produced by constructing a graph with 2^{n-1} vertices each of them labeled with one of the 2^{n-1} numbers with $n-1$ digits. The vertex labeled with $a_1a_2a_3\dots a_{n-1}$ has an outgoing edge to the vertex $a_2a_3a_4\dots a_{n-1}0$. The edge is labeled with $a_1a_2a_3\dots a_{n-1}0$. There is also an edge from $a_1a_2a_3\dots a_{n-1}$ to $a_2a_3a_4\dots a_{n-1}1$ labeled with $a_1a_2a_3\dots a_{n-1}1$. The resulting graph has an Euler circuit corresponding to a circular ordering of the possible bit sequences. Figure 5 shows the results for $n = 4$.

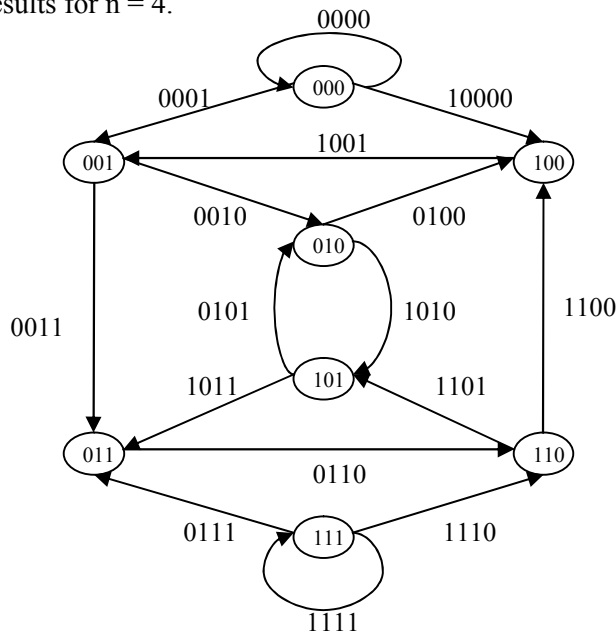


Figure 5: Euler Path $\{e_0e_1e_2e_3e_{10}e_4e_9e_3e_6e_{13}e_{11}e_7e_{15}e_{14}e_{12}e_8\}$ corresponds to the desired order of the bit sequences.

1.6 Applications

Eulerian graphs are used rather extensively, as they're necessary to solve important problems in telecommunication, parallel programming development and coding. Moreover, the corresponding theory underlies in many classic mathematical problems. In the next sections, we examine some interesting examples

Line Drawings

You can consider a line drawing as a graph whose vertices are not shown and are placed in the intersection of each pair of adjacent edges. A well-known problem is to find a unicursal tracing for a given graph.

Definition 5: *A graph has a unicursal tracing if it can be traced without lifting the pencil or retracing any line.*

Obviously, a closed unicursal tracing of a line drawing is equivalent to an Euler circuit in the corresponding graph. Similarly, an open unicursal tracing equals to an Euler path. Thus, we end up with the following conditions:

- A line drawing has a closed unicursal tracing iff it has no points of intersection of odd degree.
- A line drawing has an open unicursal tracing iff it has exactly two points of intersection of odd degree.

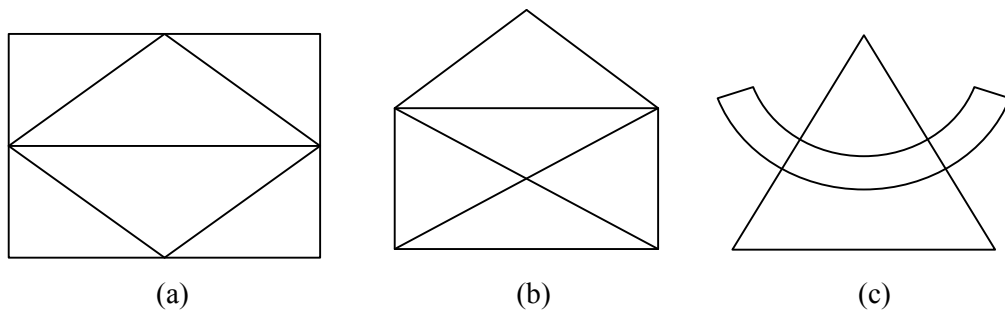


Figure 6: Shapes have an open unicursal tracing

Eulerization and Semi-Eulerization

In cases where an Eulerian circuit or path does not exist, we may be still interested of finding a circuit or path that crosses all edges with as few retraced edges as possible. Eulerization is a simple process providing a solution for this problem. Eulerization is the process of adding duplicate edges to the graph so that the resulting graph has not any vertex of odd degree (and thus contains an Euler circuit). A similar problem rises for obtaining a graph that has an Euler path. The process in this case is called Semi-Eulerization and ends with the creation of a graph that has exactly two vertices of odd degree.

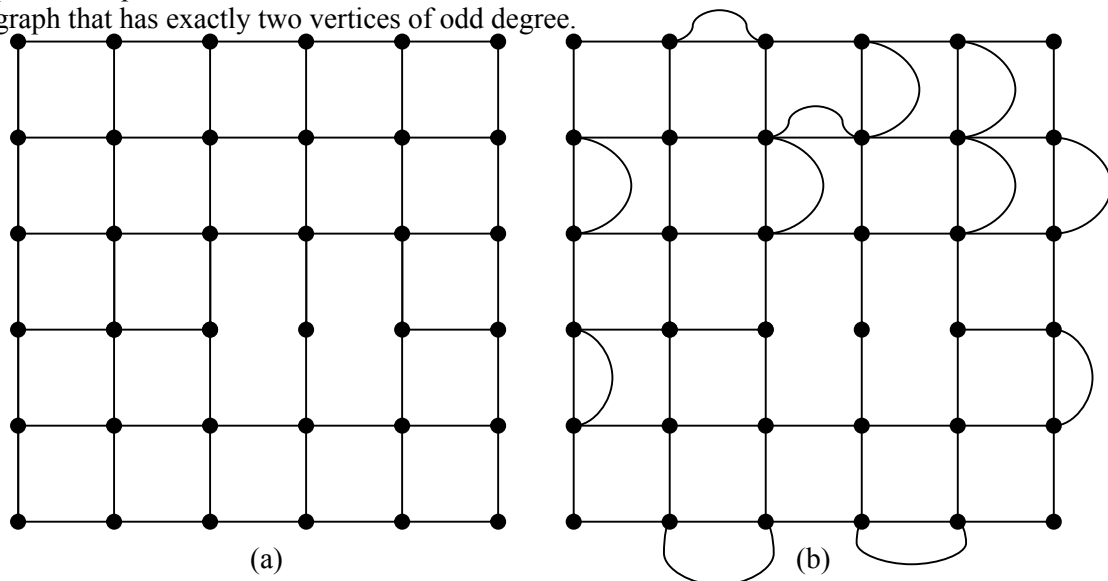


Figure 7: The initial graph (a) and the Eulerized graph (b) after adding twelve duplicate edges

Some worth mentioned points are:

- We cannot add truly new edges during the process of Eulerizing a graph. All added edges must be a duplicate of existing edges (that is, directly connecting two already adjacent vertices).
- Duplicate edges (often called “deadhead edges”) can be considered as new edges or as multiple tracings of the same edge, depending on the problem semantics.
- Eulerization can be achieved in many ways by selecting a different set of edges to duplicate. We can demand that the selected set fulfills some properties, giving birth to many interesting problems, such as the problem of the Chinese postman (Find the shortest path in a weighted graph that can have more than two vertices of odd degree).

2. The Hamiltonian Circuit problem

2.1: Introduction

A problem that at first sight seems to be very close to the Eulerian path is the Hamiltonian Circuit/Cycle problem, defined as follows:

Definition 6 *Given a (connected) graph $G(V,E)$ find a circuit that starts at a vertex of a graph, passes through every vertex exactly once, and returns to the starting vertex.*

This circuit is called Hamiltonian Circuit.

The problem can be slightly changed so that the ending vertex does not have to be the starting one. Instead of a Hamiltonian circuit we get a homonymous path.

The problem of deciding whether a graph has a Hamiltonian circuit/path (and finding one) or not is NP-complete in the general case. We will later see though that for some specific types of graphs it is solvable in polynomial time.

For weighted graphs a more important and practically useful challenge is to find a Hamiltonian path with the minimum sum of weights over the edges we pass. This path is called an optimal Hamiltonian path. The problem is well known as the Travelling Salesman Problem (TSP).

2.2: Existence

Before trying to find a Hamiltonian path or circuit it is useful to try to find whether the examined graph has one. This is an NP-Complete problem as well but for some specific types of graphs criteria that help us decide faster have been set. Here is a list of the most useful and commonly used:

Definition 7 *A directed graph $G(V,E)$ such that for each pair v_i, v_j in V either (v_i, v_j) or (exclusive) (v_j, v_i) belongs in E is called Tournament Graph*

Theorem 5 *A tournament graph always contains a Hamilton path.*

The proof is a matter of induction.

For undirected graphs we have the following, trivial theorem

Theorem 6 *(it is obvious that) any complete or chain (all nodes have degree=2) graph has Hamiltonian circuits.*

Additionally, in the last 15 years researchers have proved that any triangular planar graph can be tested in polynomial time and this resulted in a useful theorem:

Definition 8 a planar graph that will remain connected if we subtract any 4 edges, but there is a set of 5 edges which being omitted results in a non-connected graph is called 4-connected.

Tutte proved that any 4-connected planar graph contains a Hamilton circuit. This circuit can be found in polynomial time.

Theorem 7 : Any 4-connected planar graph contains a Hamilton circuit that can be found in polynomial time.

Also, Tutte proved that any graph obtained by deleting a vertex from a 4-connected planar graph contains a Hamilton circuit.

Unfortunately, in the general case we have to try to build a Hamilton path or circuit and decide that there does not exist any only after exhaustive search of the paths. But it can become even worse (Murphy's law) if we consider the fact that slight changes in the graph may result in a totally different result.

For example, we can see (via exhaustive search) that the following graph(Fig 8.1) has a Hamiltonian cycle:the path **AGFECDBA**.

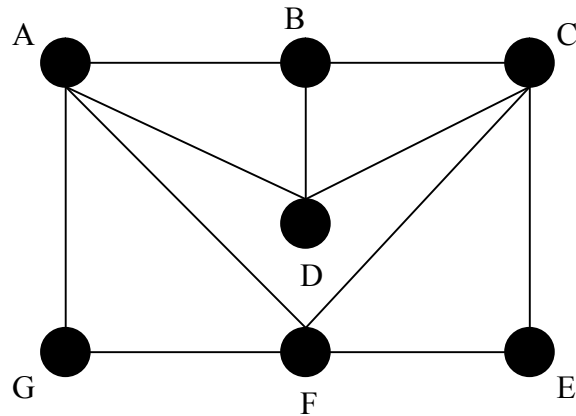


Figure 8.1: Example 1

If we remove the edge BD we have the following graph(Fig 8.2):

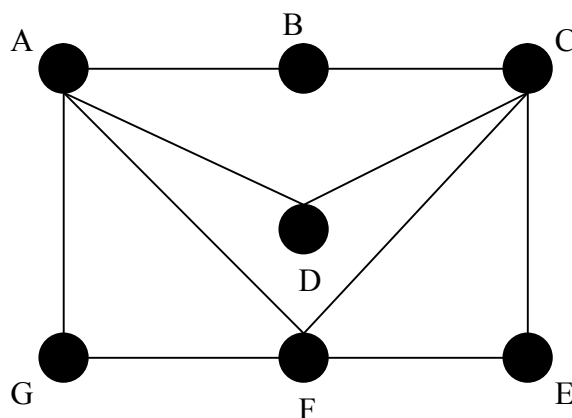


Figure 8.2 example (con.)

In this graph there is no Hamilton circuit. A sort of "proof" is the following consideration:

Any circuit can be written so that it starts at any vertex you like. So suppose you try to follow a circuit starting at vertex B. No matter whether you travel left or right, there is no way for

you to go back to B without retracing an edge unless you skip vertex D. But if you skip vertex D, then you don't have a Hamilton circuit

An encouraging note is that we can list though a number of conditions that must be held in a graph that has a Hamilton circuit:

Necessary conditions for Hamilton circuit

Assume $G=(V, E)$ has more than 2 nodes.

1. No vertex of degree 1. If node a has degree 1, then the other endpoint of the edge incident to a must be visited at least twice in any circuit of G .
2. If a node has degree 2, then both edges incident to it must be in any Hamilton circuit.
3. No smaller circuits contained in any Hamilton circuit (the start/endpoint of any smaller circuit would have to be visited twice).
4. There must exist a subgraph of G with the following properties:
 1. H contains every vertex of G
 2. H is connected
 3. H has the same number of edges as vertices
 4. H has every node with degree 2

Then , the subgraph H is the Hamilton circuit in the graph G .

The problem is that the 4th condition can not be tested in polynomial time, but the 3 first conditions can be tested fastly.

Furthermore, a very useful theorem and quite general (the only restriction is that each vertex should have $|E|/2$ adjacents or more)was proved by Dirac:

Theorem 8(Dirac's Theorem) : *If every vertex of a connected graph with 3 or more vertices is adjacent to at least half of the remaining vertices, then the graph has a Hamilton circuit..*
The proof is beyond the purpose of this passage, so it is omitted.

So we can see that even though we have a number of theorems for specific types of graphs the matter of existence of a Hamilton cycle is NP-complete in the general case.

2.3 Finding an optimal Hamiltonian circuit/path.

As said before, not only the problem is NP-complete, but we do not have any algorithm (even heuristic) to solve it in the general case as well. The only method is the exhaustive search which is "prohibited" when $|V|$ and $|E|$ become big. A more useful problem is the traveling salesman problem, which was originally set as follows:

Definition 9 (TSP) "Which is the best way for a salesman to visit a given number of cities?"

Here, best has the meaning of the minimum distance the salesman has to cover, but depending on the context given it may mean different things such as lower cost, minimum time etc.

We can represent each city by using a node, the way between any two cities as an edge between the respective nodes and the distance as the weight of the edge. The result is a complete graph, unless there is no way to go directly from one city to some other. In this case the respective edge does not exist.

The algorithm proposed is a brutal-force algorithm:

1. List all possible Hamilton circuits (of course exact reversals can be omitted)
2. Find the weight of each
3. Choose (the) one with the smallest weight.

This algorithm always works, but its main backbone is its cost.

The 1st step can only be solved by exhaustive search. What we mean is:

For a computer doing 10,000 circuits/sec, it would take about 18 seconds to handle 10 vertices, 50 days to handle 15 vertices, 2 years for 16 vertices, 193,000 years for 20 vertices!!!

This has made researchers change their approach to the problem and find approximation algorithms that may not produce the optimal result but get close to it and (more important) work in acceptable time limits.

Three such algorithms are given here: They all take as input a graph and return a close-to-the-optimal-Hamilton circuit (if one exists).

CHEAPEST LINK ALGORITHM (CLA)

1. Choose the edge with the smallest weight (the "cheapest" edge), randomly breaking ties
2. Keep choosing the "cheapest" edge unless it (a) closes a smaller circuit OR (b) results in 3 selected edges coming out of a single vertex

Continue until the Hamilton Circuit is complete

Figure 9.a below shows a graph (K5) on which we apply this algorithm: We choose AC (min weight). Then, since no restriction will be validated we add AD. Now if we try to add DC we get a cycle ACDA (Fig. 9.b) so we do not add it. Instead we take the next lower weighted edge DB and add it. If we add ED node D will have degree 3 (Fig 9.c), so we do not use it. Instead we add EC and EB successively so we get a Hamiltonian cycle ADBECA (Fig 9.d) with total cost 20

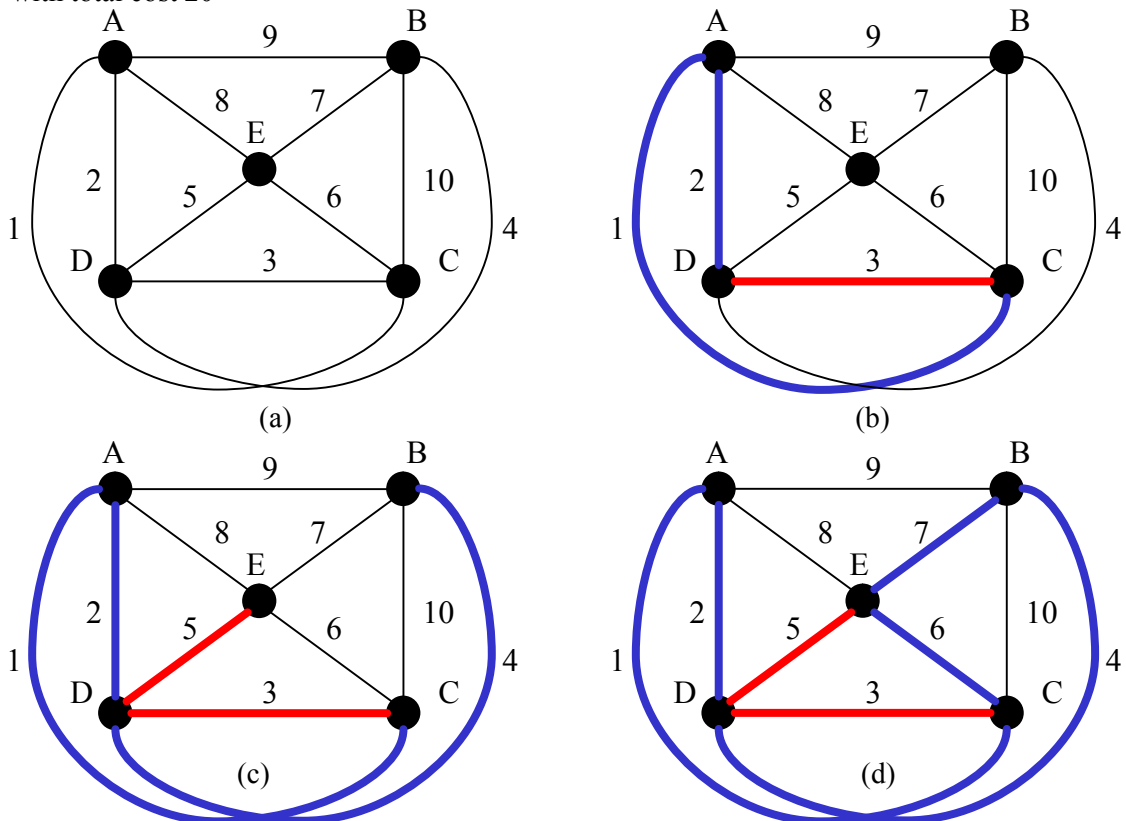


Figure 9: Applying CLA algorithm

NEAREST NEIGHBOR ALGORITHM (NNA)

1. Start at a vertex (think of it as your Home city)
2. Travel to the vertex that you haven't been to yet whose path has the smallest weight
If there is a tie, pick randomly.
3. Continue until you travel to all vertices
4. Travel back to your starting vertex

In figure 4 we show how this algorithm is applied for the graph of the figure 2.3.a

Randomly we choose E as the initial vertex. We choose the edge with the lower cost starting from E (Fig10.a) which is ED. Same way we add DA and AC and reach 10.b. Here we can not use the edge with the lower cost (CD) since D has already been visited so we travel to B through CB (Fig 10.c) and finally reach the circuit EDACBE (Fig 10.d) with total cost 25 (worse than the previous)

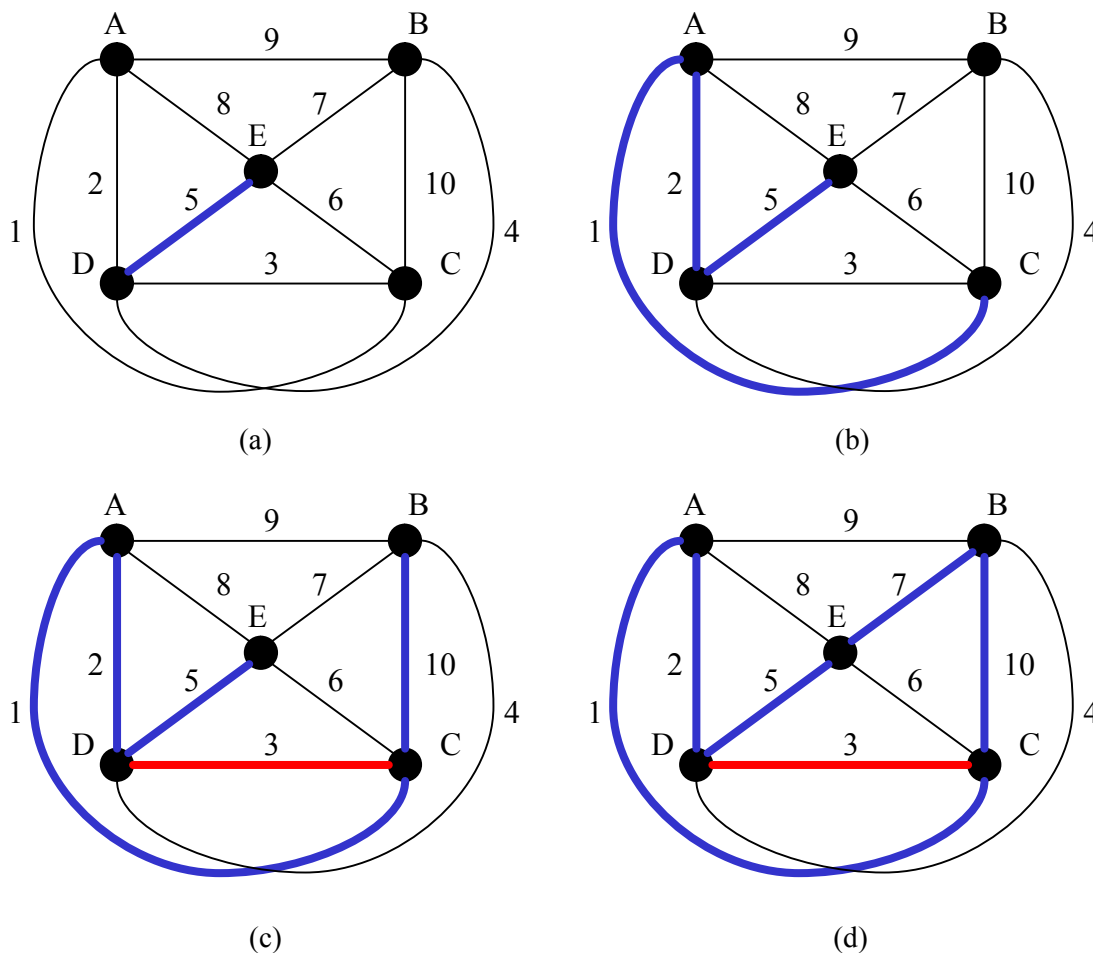


Figure 10: Applying NNA

REPETITIVE NEAREST NEIGHBOR ALGORITHM (RNNA)

1. DO NNA for each vertex (city)
2. Choose the best solution (smallest weight)
3. If necessary, rewrite this solution with a particular starting vertex (Home)

It uses the previous algorithm, which is depended on the choise of the initial node
That's why we apply it for every initial node possible and choose the best circuit among the ones produced.

Definition 2.4 *The criterion to decide which is the more efficient is the relative error: $R_e = (\text{difference from optimal solution}) / (\text{optimal solution})$*

A similar problem is to find a Hamiltonian path given the initial and final nodes. This would help the salesman end his journey at his home and not the city he was at the beginning!!!

There exists a constructive algorithm that gives us the optimal Hamilton path between two nodes a (initial) and p (terminal). This algorithm tries to find the optimal path between any two vertices visiting any number ($\leq |V|-2$) of vertices starting from 0 (no intermediates) and adding one node to the (new) path at the time.

E.g in order to find the minimum path between a and p that has 3 intermediate nodes we find all the paths from any node k to p with 2 intermediate nodes ($\neq a, p$) and add (a,k) and choose the minimum.

Optimal Hamilton Path

Input : w: matrix of weights, a: initial node, p: final node and $N=|V|$

Output: The optimal Hamilton Path

optimal Hamilton Path(w,a,p)

S: subset $\{1,2,\dots,N\}$

Array H [N] [N] [S]

int i,j,s,k

1. for $i,j=1$ to N
2. $H[i,j,\emptyset]=w(i,j)$
3. for $s=1$ to N-2
4. for $i,j=1$ to N
5. for each $S \subseteq \{1 \dots N\} - \{i,j\}$ with size s
6. $H[i,j,S]=\min_{k \in S} [d(i,k)+H(k,j,S-\{k\})]$
7. return $H[a,p,\{1 \dots N\} - \{a,p\}]$

The algorithm runs in $T(2^N)$ time significantly lower than $T(N!)$ which gives us the ability to work with larger graphs in appropriate time. The reason we have this complexity has to do with the number of the paths we have to calculate in step 6 before we select the minimum among them.

3. Comparison of Euler and Hamilton Circuits

At this point it must be obvious that the similarity of the two problems discussed is just illusionary. There are many differences both in their practical appliances and their theoretical analysis. First of all the Eulerian problem takes edges in consideration while the Hamiltonian one has to do with vertices. The first one tries to pass through every edge of the graph exactly once, while the second one tries to visit all the nodes exactly once without retracing any edge. The nature of the problems shows that in the first case we are not interested whether the graph is weighted or not, while the second one has much bigger practical use if the graph is weighted.

The analysis showed that despite their “similarity” the Eulerian problem can be solved in linear time while the question whether a Hamilton cycle exists or not can only be answered in NP-time in the general case, $O(N!)$ in fact!! Here is a list with the most important differences between the two problems.

Euler Circuit

Repeated visits to a given node are allowed.

Traverse each edge exactly once (by definition of problem).

No node may be omitted (if so, one of the edges incident to the node would not be traversed).

No edges may be omitted (by definition of the problem)

Hamilton Circuit

Visit each node exactly once.

Repeated traversals of a given edge are not permitted, since that would result in visiting a node more than once.

No node may be omitted (by definition of the problem).

Any edge may be omitted, as long as its endpoints can be reached via some other path.