Chapter 10

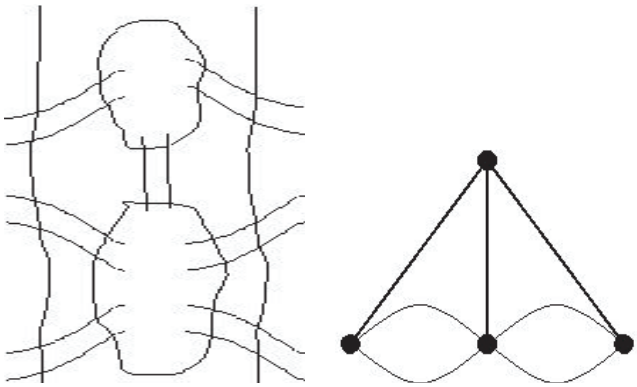
Eulerian and Hamiltonian Paths Circuits

This chapter presents two well-known problems. Each of them asks for a special kind of path in a graph. If there exists such a path we would also like an algorithm to find it. Both of the types of paths (Eulerian and Hamiltonian) have many applications in a number of different fields. The chapter examines these two problems. After this, the Travelling Salesman Problem (TSP), another problem with great practical importance which has to do with circuits will be examined.

10.1 Euler paths and circuits

10.1.1 The Konisberg Bridge Problem

Konisberg 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 10.1. 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.



(a) Konisberg bridge (b) respective graph

Figure 10.1: Konisberg bridge and the graph induced

10.1.2 Defining Euler Paths

Obviously, the problem is equivalent with that of finding a path in the graph of figure 10.1(b) such that it crosses each edge exactly once. 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 have his name.

Definition 10.1 *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 10.2 *The degree of a vertex u in a graph equals to the number of edges attached to vertex u.*

10.1.3 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 following theorem which sets a sufficient and necessary condition for the existence of an Euler circuit or path in a graph.

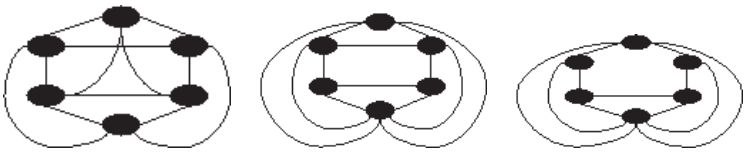
Theorem 10.1 ***(Euler's theorem)***
An undirected graph has at least one Euler circle iff it is connected and has no vertices of odd degree. An Euler path exists exist iff there are no or zero vertices of odd degree.

Proof.
$\Rightarrow$: An Euler circuit exists. As the respective path is traversed, each time we visit a vertex there is a vertex through an edge we can leave through another edge. For the starting-finishing vertex, this also holds, since there is one edge we initially leave from and another edge, through which we form the circle. Thus, whenever we visit a node we use two edges, which means that all vertices have even degrees.

$\Leftarrow$: By induction on the number of vertices.
Induction beginning: $|V| = 2$, trivial.
Induction basis: Suppose for $|V| = n$ the theorem holds.
Induction step: Show that the theorem holds for $|V| = n + 1$.
Select an arbitrary vertex u and build a graph of size with n vertices by removing this vertex and all the edges adjacent to it. Vertex u had even degree so it has an even number of neighbors. Now all the neighbors of v have odd degree since one adjacent edge is removed. By grouping the neighbors in couples and adding one edge between each couple, we obtain a graph with n vertices, where every vertex has even degree. Thus, there exists an euler circuit (induction basis). When an edge (u,w) added between neighbors of v is met while traversing the circuit, we can replace it by the path $(u, v) - (v, w)$. This way every edge in the graph initial is traversed exactly once, so there exists an eulerian circuit.

In case we have two vertices with odd degree, we can add an edge between them, obtaining a graph with no odd-degree vertices. This graph has an euler circuit. By removing the added edge from the circuit, we have a path that goes through every edge in the graph, since the circuit was eulerian. Thus the graph has an euler path and the theorem is proved.

Example: Figure 10.2 shows some graphs indicating the distinct cases examined by the preceding theorem. The graph in fig 10.2(a) has an euler circuit, in fig 10.2(b) the graph has an euler path but not an euler circuit and in the graph of fig 10.2(c) there is neither a circuit nor a path.



(a) Graph with euler circuit

(b) Graph with euler path

(c) Graph with neither euler circuit nor path

Figure 10.2: Examples of graphs

10.1.4 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 be solved intuitively respecting a few guidelines:

1. 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.

2. 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).

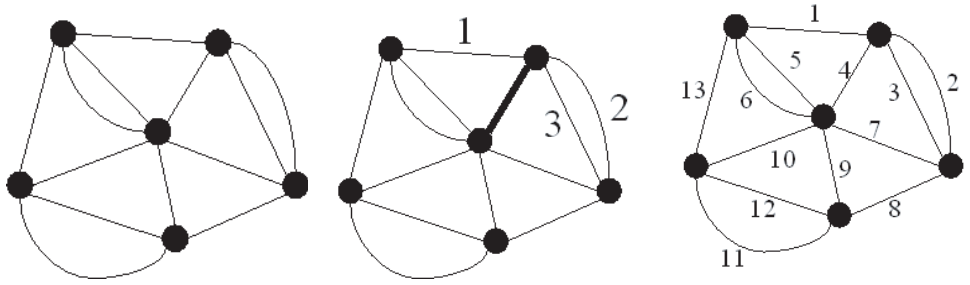
Two constructive algorithms for obtaining an euler circuit/path are presented here:

FLEURY'S ALGORITHM(G(V,E))

1 choose some vertex u_0 of G
2 $P = u_0$
3 consider $P = u_0e_1u_1e_2...e_iu_i$ and choose an edge e_{i+1} with the following properties
4 (1) e_{i+1} joins u_i with some vertex u_{i+1} and
5 (2) the removal of e_{i+1} does not disconnect the graph if possible
6 add e_{i+1} and u_{i+1} in the path
7 remove e_{i+1}
8 **if** $|P| = W$
9 **then** return P
10 **else**
11 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 two vertices of odd degree as the beginning vertex. The final vertex of the path will be the other odd-degree vertex.

Example: Figure 10.3 demonstrates some important steps in the process described by the algorithm. Since all vertices have odd degree we arbitrarily start from the upper left vertex. The number next to each edge indicates its order in the Euler circuit.



(a) The graph (b) Can not avoid crossing a bridge (c) Full path

Figure 10.3: Example of fleury's algorithm execution

Eulerian graphs, have a very important property: They consist of smaller rings. Rings are cycles with the additional restriction that during the traversal of the cycle no vertex is visited twice. Let us consider an eulerian graph, we know that every vertex has an even degree. Any ring passes through exactly two edges adjacent to any of its nodes. This means that if we remove the ring, the remaining of the graph has still an even degree for all of its nodes, thus remains eulerian. By repeating this procedure until no edges are left we can obtain a decomposition of an eulerian graph into rings. Of course, we can decompose an eulerian cycle to smaller cycles, not necessarily rings, but rings have a higher practical value, in terms of networks. This is of high importance in network design, where we want to keep a network alive even when a number of links are down.

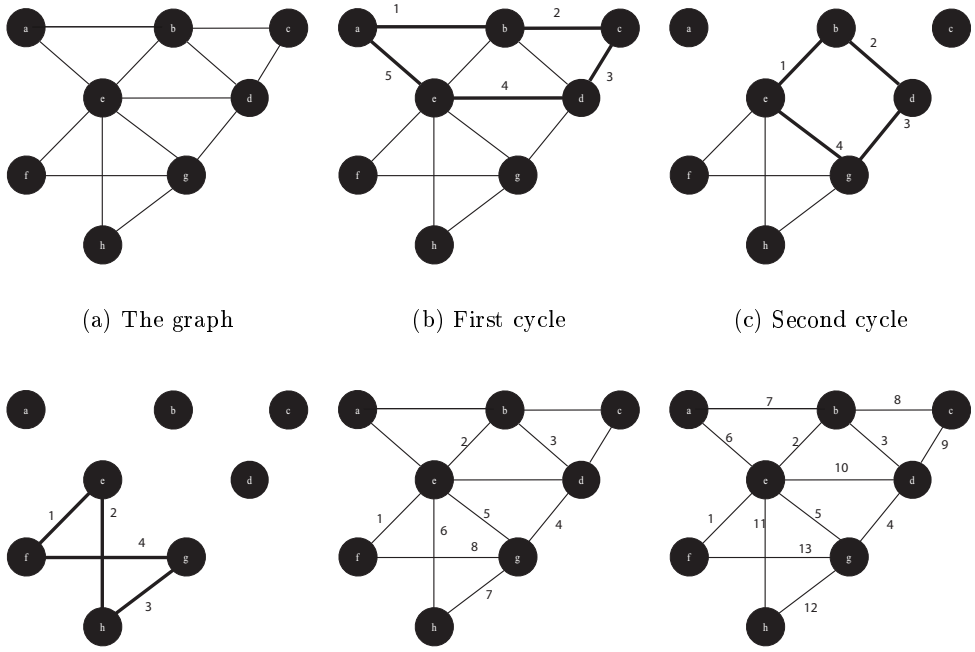
This property can also help as build the eulerian circle with the aid of the small rings, or cycles a graph can be decomposed to. The procedure we follow is described here

A CONSTRUCTIVE ALGORITHM FOR BUILDING EULERIAN CIRCUITS(G(V,E))

```
1   choose some vertex u_0 of G
2   start travelling through edges not visited yet until a cycle is formed.
3    record the cycle and remove the edges it consists of
4  if  there are unvisited edges
5     then
6              goto 1
7     else
8              merge the recorded cycles
9              return
```

Two cycles $C_0 = u_0\ u_1\ ...\ u_i\ u_0$ and $C_1 = v_0\ v_1\ ...\ v_j\ v_0$ are merged by traversing one of them and insert the other when a common vertex is found. The result is a new cycle.

The computational complexity of this algorithm is O(E), since we only traverse edges until we form a cycle. For the merging procedure O(E) time suffices since once again we



(a) The graph (b) First cycle (c) Second cycle

(d) Third cycle (e) Amalgamating second and third cycle (f) Amalgamating with the first cycle: euler cycle

Figure 10.4: Example of the constructive algorithm

traverse the set of edges.

We can easily make this algorithm find euler paths, using the same trick as in Euler's theorem's proof. There must exist exactly two vertices with odd degree, otherwise no Euler path can be found. We add an edge between these two vertices, compute an euler circuit, add obtain the path by removing the added edge.

Example: Figure 10.4 demonstrates the process described by the algorithm. There are 3 different edge-disjoint cycles identified: $a \to b \to c \to d \to e \to a$ (in fig. 10.4(b)), $e \to b \to d \to g \to e$ (in fig 10.4(c))and $f \to e \to h \to g \to f$ (in fig 10.4(d)). We can amalgamate the two later cycles to obtain a bigger circle: $f \to e \to b \to d \to g \to e \to h \to g \to f$ (in fig 10.4(e)). Then this cycle is combined with the first one giving $f \to e \to b \to d \to g \to e \to a \to b \to c \to d \to e \to h \to g \to f$, which is an Euler cycle (fig 10.4(f)).

10.1.5 Expansion to directed graphs

Expanding to directed graphs is quite straightforward. As before, it is obvious that if an euler circuit exists, during its traversal, one must always visit and leave every vertex. This means that the number of edges leading to a vertex (in-degree) must be equal to the number of the edges that leave the vertex (out-degree). This time, the condition for the existence of a path is slightly different, since for the first vertex of our path v we have $in - degree(v) = out - degree(v) - 1$ and for the last vertex u $in - degree(u) = out - degree(u) + 1$. That is

because we start from the first vertex using an out-going edge and finish at the final vertex through an in-coming edge. So for directed graphs the following theorem stands.

Theorem 10.2 *A directed graph has at least one Euler circle iff it is connected and for every vertex u in-degree(u)= out-degree(u). An Euler path exists exist iff there are exactly two vertices s,f for which the previous criterion does not hold and for which in-degree(s)= out-degree(s) -1 (starting vertex of the path) and in-degree(f)= out-degree(f)+1 (final vertex of the path).*

The euler circuits and paths can be obtained using the same algorithms as before, only this time the direction of an edge during its traversal must be taken into consideration.

10.1.6 Applications

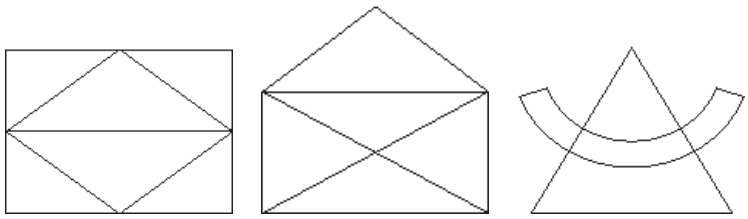
Eulerian graphs are used rather extensively, as they're necessary to solve important problems in telecommunication, parallel programming development and coding. Moreover, the corresponding theory underlies in many classic mathematical problems. In the next sections, we examine some interesting examples

Line Drawings

This is a mathematical game, where given a shape (line drawing) one is asked to reproduce it without lifting the pencil or retracing a line. 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.

Definition 10.3 *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". In figure 10.5 such drawings appear.



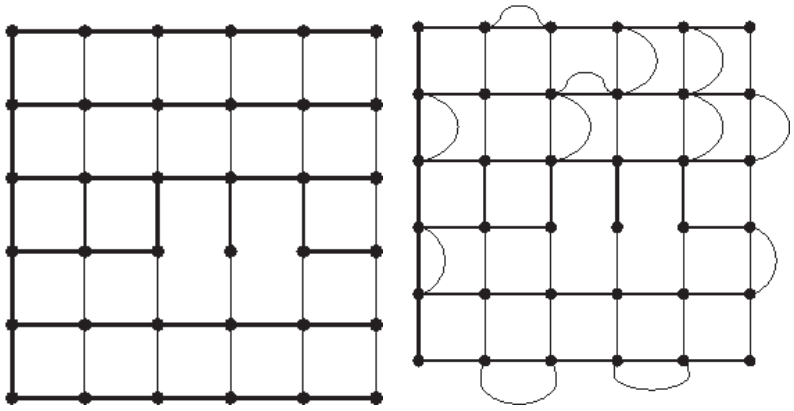
(a) open (b) open (c) closed

Figure 10.5: Unicursal tracing

10.1.7 Eulerization and semi-Eulerization

In cases where an Eulerian circuit or path does not exist, we may be still interested in 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). We can do this by selecting pairs of vertices with odd degree and duplicating the edges that form a path between them. For any intermediate vertex we add(duplicate) two edges keeping its degree even if it was even and odd if it was odd. At this point we must recall the property of any graph that the number of vertices with odd degree is even. This means that no odd-degree vertex remains uncoupled. An example of a non-eulerian graph and its eulerization appears in figure 10.6

A similar problem rises for obtaining a graph that has an Euler path. The process in this case is called Semi-Eulerization and is the same as before with the only addition that we add edges in such a way that the initial and final vertices of the path have odd degree. This means that if the vertex we want the path to start from (or end to) has even degree we have to duplicate some edges so the degree becomes odd.



(a) a non- eulerian graph (b) Eulerization of the graph

Figure 10.6: Eulerization process

Some worth mentioned points are:

1. 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).

2. 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.

3. 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 asking for the minimum number of edges to be duplicated

10.2 Hamilton paths and circuits

Another important problem having to do with circuits and paths is the search for a cycle that passes through every vertex exactly once. This means that not all edges need to be traversed. Such cycles, and the respective paths (that go through every vertex exactly once) are called Hamilton circuits/path and graphs that contain hamilton circuits, are characterized as hamiltonian.

Definition 10.4 *A hamiltonian circuit is a circuit that starting from a vertex u_0 passes through all other vertices u_i exactly once and returns to the starting vertex. A hamiltonian path similarly is a path that starting from a vertex u_0 passes through all other vertices u_i exactly once and stops at a final vertex.*

The problem of finding a hamilton circuit or path, is an NP-complete problem, thus it is highly unexpected to find a polynomial algorithm for solving it. There exist however several criteria that determine whether a graph is hamiltonian or not for some families of graphs.

Unfortunately, global assumption such as high density, or a guaranteed minimum degree are not enough. We can easily construct a non- Hamiltonian graph whose nodes' minimum degree exceeds any given constant. What if we use a variable instead of a constant? Dirac stated and proved the following theorem:

Theorem 10.3 ***(Dirac 1952)***
Every graph with $n \geqq 3$ vertices and minimum degree at least $n/2$ has a Hamilton cycle.

Proof.
Let G(V,E) be a graph with $|N| \geqq 3$ and $\delta(G) \geqq n/2$. First of all, the graph is connected, otherwise the degree of every vertex in the smaller component C would be less than $|C| = n/2$

Let $P = x_0...x_k$ be a longest path in G. This means that all neighbors of x_0 and x_k lie on P. Otherwise, the path could be increased by adding a not already included neighbor, which contradicts the maximality of P. Hence, at least $n/2$ of the vertices $x_0x_1...x_{k-1}$ are adjacent to x_k and at least $n/2$ of the same vertices x_i for which x_{i+1} are neighbors of x_0. Since the two sets have at least $n/2 + n/2 = n$ vertices and the longest path can not have more than n vertices there is a vertex x_i that is adjacent to x_k and for which x_{i+1} is a neighbor of x_0 (figure 10.7). Then the cycle $C = x_0 \to x_{i+1} \to P(x_{i+1} : x_k) \to x_i \to P(x_i : x_0)$ forms a hamilton cycle. That is because no vertices exist that are not included in C. If there was one such vertex, it would have to be connected to a vertex in C since the graph is connected. This would lead in a larger path than P, which is a contradiction to our hypothesis that P is a longest path.

Another theorem is based on the independence number $a(G)$ of a graph G.

Definition 10.5 *An independence set V' of a graph $G(V, E)$ is subset $V' \subseteqq V$ for which holds: For any two vertices u, v of V' (u, v) is not an edge in G*

Definition 10.6 *The independece number $a(G)$ of a graph $G(V, E)$ is the cardinality of the largest independence set of G*

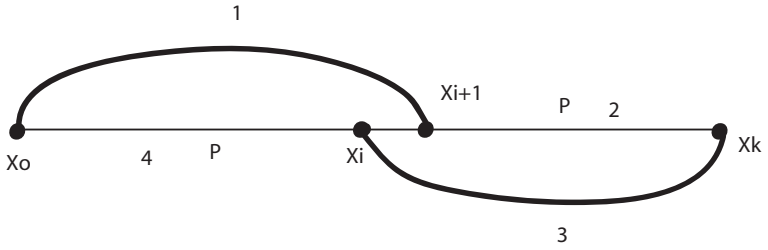


Figure 10.7: Hamilton cycle

For this theorem we also need the definition of $k(G)$. $k(G)$ is the largest integer k for which G is k-connected.

Now, with the definition of *independence number* given, we can proceed and introduce the theorem.

Theorem 10.4 *Every graph with $n \geqq 3$ vertices and $k(G) \geqq a(g)$ has a Hamilton cycle.*

Proof.
Let k(G)=k and C be a longest cycle in G. We will show by contradiction that C has to be a hamilton cycle, so let C not be Hamilton. First, we enumerate the vertices in C cyclically e.g. $u_1u_2...u_last$ and $1, 2, ...last = Z_n$ The enumeration is cyclical which means that (u_i, u_{i+1}) is an edge of C. Now since C is not Hamiltonian, we can select a vertex u from $G - C$, and create a fan $u - C$ named $F = P_i : i \in I$ where $I \subseteqq Z_n$ and $P_iisanedgefromutou_i$. If we select F to be of maximum cardinality this would mean that for any j not in I (u, u_j) is not an edge. Then $|F| >= min(k, |C|)$. For every i in I we know that $i + 1$ is not in I, otherwise $C \bigcup P_i \bigcup P_{i+1} - u_iu_{i+1}$ would be a cycle larger than C, which contradicts our hypothesis (fig 10.8(a)). This means that at least one vertex of the cycle doesn't belong in the fan so $|F| \leqq |C|$ thus $|F| \geqq k$Furthermore, for all i, j in I $u_{i+1}u_{j+1}$ is not an edge, otherwise we would have $C \bigcup P_i \bigcup P_{i+1} + u_{i+1}u_{j+1} - u_iu_{i+1} - u_ju_{j+1}$ to be a cycle larger than C (fig 10.8(b)), which is against our hypothesis. So the set $\{u_{i+1} : i \in I\} \bigcup \{u\}$ is a set of at least $k + 1$ independent vertices in G. This contradicts the fact that the independence number is k. Hence, C is a Hamilton cycle.

It was a great surprise when in 1956 Tutte proved the following theorem, which is the best possible weakening.

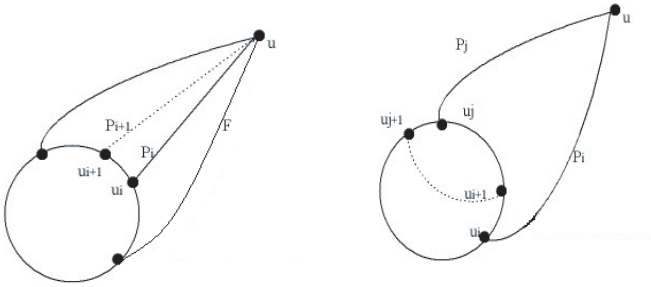
Theorem 10.5 ***(Tutte 1956)***
Every 4-connected planar graph has a Hamilton cycle.

Later, in 1972, Chvatal took into consideration the degrees of all nodes, and proved another powerful theorem. In order to continue with this theorem a few more definitions are required.

Definition 10.7 *If $G(V, E)$ is a graph with n vertices and degrees $d_1 \leqq d_2 \leqq d_3 \leqq ... \leqq d_n$ then the n- tuple $d_1d_2d_3...d_n$ is called the degree sequence of the graph*

Definition 10.8 *An arbitrary integer sequence $a = (a_1a_2...a_n)$ is called hamiltonian if every graph with n vertices and a degree sequence pointwise greater than a is hamiltonian.*

Definition 10.9 *An integer sequence $d = (d_1d_2...d_n)$ is pointwise greater than an integer sequence $a = (a_1a_2...a_n)$ if $d_i \geqq a_i$ for every i.*



(a) case 1 (b) case 2

Figure 10.8: Images for the proof of theorem 10.4

With these definitions in mind we can proceed to the following theorem.

Theorem 10.6 ***(Chvatal 1972)***
An integer sequence $a = (a_1 a_2 ... a_n)$ *such that* $0 \leqq a_1 \leqq a_2 \leqq ... \leqq a_n < n$ *and* $n \geqq 3$ *is hamiltonian if and only if the following holds for every* $i < n/2$

$$a_i \geqq \bigvee a_{n-i} \geqq n - i \tag{10.1}$$

which is equivalent to

$$a_i \leqq i \Rightarrow a_{n-i} \geqq n - i \tag{10.2}$$

Proof.

Let $a = (a_1 a_2 ... a_n)$ be an arbitrary integer sequence such that $0 \leqq a_1 \leqq a_2 \leqq ... \leqq a_n < n$ and $n \geqq 3$
$\Rightarrow$ We assume that this sequence satisfies the conditions of the theorem. We will show that it is hamiltonian using contradiction. So there must exist at least one graph $G(V, E)$ with $|V| = n$ and degree sequence d pointwise greater than a:

$$d_i \geqq a_i \forall i \tag{10.3}$$

and G is not Hamiltonian. Among these graphs we select one with maximum number of edges and enumerate its vertices $(u_1 u_2 ... u_n)$ so that $d(u_i) = d_i$ Then equation 10.1 is transformed as follows

$$d_i \geqq i \bigvee d_{n-i} \geqq n - i \forall i < n/2 \tag{10.4}$$

Next we select two distinct vertices x,y that are not adjacent and such that $d(x) \leqq d(y)$ and $d(x) + d(y)$ is as large as possible. If we add the edge (x, y) forming a new graph G_{new} it is obvious that this graph has more edges than the initial and is pointwise greater than a. Since G was selected to be the graph with maximum number of edges which is

not hamiltonian and pointwise greater than a, G_{new} has a hamilton cycle. But this means that G has a hamilton path $x_1x_2...x_n$ and let $x = x_1$ and $y = x_n$. Like we did for Dirac's theorem, we now consider the indices of vertices adjacent to x and y:
$I = \{i : xx_{i+1} \in E\}$ and $J = \{j : x_jy \in E\}$ We have $I \bigcup J \subseteq \{1, 2, ..., n-1\}$ and $I \bigcap J = \oslash$ since G does not have a hamilton circuit. This means that

$$d(x) + d(y) = |I| + |J| < n \tag{10.5}$$

so we have that $h = d(x) < n/2$ since $d(y) > d(x)$ by our choice. For all vertices adjacent to x, we know that they are not adjacent to y. Their population is h. The fact that they were not chosen instead of x means that they all have degree lower than d_h. So there are at least h vertices with degree lower than h and consequently $d_h \leqq h$. Then, equation 10.4 means that $d_{n-h} > n - h$, since $h < n/2$. Since the degrees are non-decreasingly ordered all vertices enumerated with numbers greater than $n - h$ have degree greater than $n - h$. Their population is h+1 which means that at least one of them, call it u is not adjacent to x. But then $d(u) + d(x) >= n - h + h = n$ and u,x are not neighbors. This is against our choice of x and y. Thus the graph must be hamiltonian.

$\Leftarrow$ Now we will show that for every ordered integer sequence $a = (a_1a_2...a_n)$ with $a_h \leqq h$ and $a_{n-h} \leqq n - h - 1$ for some h there exist a graph whose degree sequence is greater but it is not hamilton. Given h, it suffices to show that this holds for the pointwise greatest sequence. According to the limitation this is

$$\underbrace{h, h, ..., h}_{h}, \underbrace{n-h-1, n-h-1, ..., n-h-1}_{n-2h}, \underbrace{n-1, n-1, ..., n-1}_{h} \tag{10.6}$$

Let us test the graph which is a union of a $K_{h,h}$ with a K_{n-h}, where k_{n-h} consists of the vertices labelled $u_{h+1}...u_n$ and the two partitions of $K_{h,h}$ are vertices $(u_1u_2...u_h)$ and $(u_{n-h+1}...u_n)$. There are h vertices, namely u_1 to u_h which have exactly h neighbors, vertices $u_{n-h+1}...u_h$. There are $n - 2h$ vertices u_{h+1} to u_{n-h+1} that are adjacent to n-h-1 vertices: $u_{h+1}...u_n$. Finally there are h vertices $u_{n-h+1}...u_n$ each of which is adjacent to all n-1 vertices. Figure 10.9 shows this graph. Now the proof of the theorem is completed.

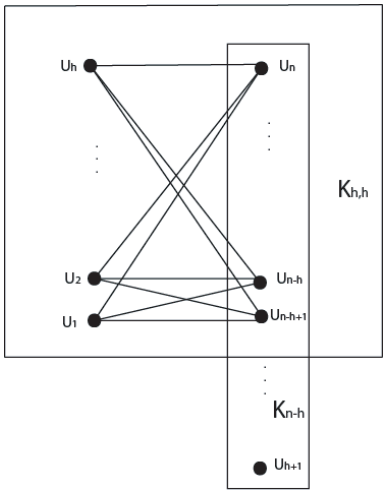


Figure 10.9: A graph that satisfies the limitations and is not Hamiltonian

Up to this point, we have seen a number of criteria that allow one to determine whether a graph is hamiltonian or not for some types of graphs. But even if we do know that a graph has a hamilton path or circuit we do not yet know how to obtain it. Once again, there are some types of graphs for which a hamiltonian path/ circuit can be obtained in polynomial time (4-connected planar graphs are such a type of graphs). But in the general case of a graph whose structure is unknown the only way to determine the existence of a hamilton path/circuit is to exhaustively search all different paths. Since it is also wanted to present a hamilton path - if one exists most algorithms try to build such a path by selecting a node and increasing a path until it becomes hamiltonian or we reach a dead-lock. Since the computational cost is extremely high during this procedure we would like to save as much time as possible by detecting paths that can not grow up to hamiltonian as early as possible. For this purpose there are several criteria set. The straightforward algorithm that searches exhaustively for hamilton circuits appears here and is followed by a set of simple criteria provided by Hakimi in 1966 and enriched by Rubin in 1974.

```
findHamiltonCircuit(G(V,E))
 1   Select any single node as the initial path
 2  if  the path is admissible AND there are unvisited nodes
 3     then
 4             list the successors of the last node chosen
 5              extend the path to the first unvisited successor
 6              goto step 2
 7
 8     else
 9             delete the last node chosen from the path
10             choose the next listed unvisited successor of the preceding node
11             goto step 2
12  if  all extensions from a given node have been shown inadmissible
13     then
14             goto step 9
15
16  if   all extensions from the initial node have been shown inadmissible
17     then
18             no circuit exists
19  if   all nodes are included in the path and the last node is adjacent to the initial node
20     then
21             a hamilton circuit exists
```

For the test for admissibility in step 2 we need to consider the following: During constructing the path we can find three types of edges.

1. **Required:** If a graph has exactly two edges incident to it both of the edges are required in order to visit and leave during traversing a hamilton circuit.

2. **Deleted:** If a node is already in the path then no other edges incident to it can be used during traversing a hamilton circuit. Furthermore, any edge that closes a circuit other than hamiltonian can not be used. These edges can be deleted.

3. **Undecided:** All edges that are not in any of the two previous categories can potentially be used some time later in order to increase the path.

Whenever adding a vertex in the path we need to update all these categories. With this classification we can detect paths that can not increase to hamiltonian following using the following set of rules.

1. Fail if any vertex becomes isolated. Since it is isolated it cannot be reached thus a hamilton circuit can not be constructed
2. Fail if any vertex has only one incident arc. This vertex can be reached but then we can not leave following a different edge
3. Fail if any vertex has three required arcs incident. In any circuit every vertex is incident to exactly two edges
4. Fail if any set of required arcs forms a closed circuit, other than a Hamilton. A Hamilton circuit can not contain any smaller circuits

Furthermore we can take into consideration that connectivity alters while edges are removed. This means that two more rules for failure can be added

1. Fail if for any vertex not already included in the path there is no path to the initial vertex
2. Fail if there exists a vertex not already included in the path which is unreachable from the last vertex added in the path.

Of course the list of rules is not complete. Many more criteria exist, others simple, like those mentioned here and others more sophisticated. In any case one must keep in mind that the set of rules used has to be fast to apply, otherwise the time gained by not searching doomed to fail paths will be lost during checking.

10.2.1 Hamilton paths and circuits in directed graphs

In undirected graphs, it is obvious that a complete graph (a clique), where all possible edges exist is hamiltonian. What is important is that we can direct the edges of a complete graph in any way we want and the obtained directed graph will always contain a hamilton path. Such graphs are called tournaments.

Definition 10.10 *A directed graph $G(V, E)$ such that for every pair of vertices u, v either $\overrightarrow{(u, v)} \in E$ or $\overrightarrow{(v, u)} \in E$ is called tournament.*

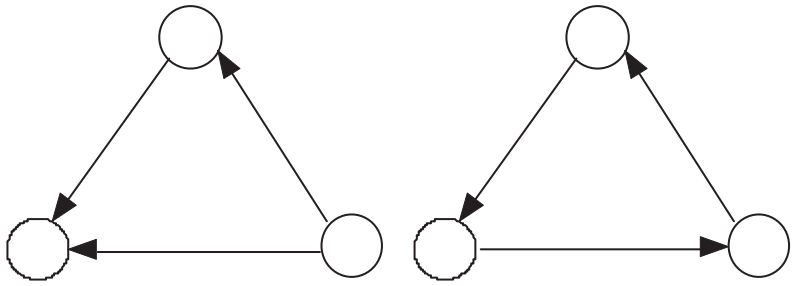
As already mentioned a tournament graph always contains a hamilton path.

Theorem 10.7 *Any tournament graph contains a hamilton path*

Proof.

The proof is by induction on the number of vertices. For a directed graph with three vertices one of the two cases shown in figure 10.10 can appear. Both of these have a hamilton path. All other directed graphs with three vertices are isomorphic to one of the two

cases. The hypothesis is that any tournament graph with k vertices has a hamilton path. We want to show that a tournament with $k+1$ vertices has a hamilton path. So we consider a $k+1$ tournament graph $g(V, E)$ and select randomly a vertex y. The remaining of the graph $G - y$ has k vertices and is tournament so it is has a hamilton path, call it $P = \overrightarrow{x_1 x_2 ... x_k}$. For every x_i either $\overrightarrow{x_i, y}$ or $\overrightarrow{y, x_i}$ exists. If $\overrightarrow{y, x_1}$ exists then G has a hamilton path $y \to x_1 \to x_2 ... \to x_k$ Otherwise $\overrightarrow{x_1, y}$ exists. In this case we traverse the path $x_1 \to\to ... \to x_k$ until an edge $\overrightarrow{y, x_j}$ rather than $\overrightarrow{x_j, y}$ is found for the first time. The $x_1 \to x_2 \to ... \to x_{j-1} \to y \to x_j \to ... \to x_k$ is a hamilton path for the graph G If no such vertex x_j exists, then all edges involving y are directed towards it. This means edge $\overrightarrow{x_k, y}$ exists. Then $x_1 \to ... \to x_k \to y$ is a hamilton path for G. So in any case G contains a hamilton path, thus theorem is proved.



(a) case 1 (b) case 2

Figure 10.10: Directed graphs with 3 vertices

Once again the problem is not to simply determine whether a graph is Hamiltonian or not. We would also like to detect a hamilton circuit if one exists. Unfortunately like in the case of undirected graphs, even if the problem is solvable in polynomial time for some types of graphs, in the general case only exhaustive search can guarantee correctness. The exhaustive algorithm described earlier can be used for detecting hamilton circuits in directed graphs with slight modifications and additions.

10.2.2 Comparison to the Euler circuit problem

We have examined two different problems with a common subject. In both cases we are interested in finding a circuit. When asking for an Euler circuit we want to find a circuit that contains every edge of a graph exactly once. This is of importance when we want to take advantage of all the edges, e.g if we want to have a network that can survive if a number of links is down. On the other hand when asking for a Hamilton circuit we want to travel through all the vertices of a graph, without being interested in the edges we use. We are interested in such graphs when we want to travel over all vertices quickly, without using an edge two or more times. These two problems are very different. An Euler circuit can be obtained in $O(E)$, proportional to the number of edges time. On the contrary, we can not find a Hamilton circuit, or even determine its existence without exhaustively searching the graph. There are some categories of graphs for which we have polynomial algorithms, but in general this is an NP-complete problem. Furthermore, assigning weights on the edges of the graph plays no role when searching for an Euler circuit: All edges have to be visited, so

the total cost will always be the same, namely the sum of the weights of all edges. But in the case of Hamilton circuits, having a weighted graph is a much more challenging problem: Not only we are looking for a Hamilton circuit, but we want the circuit to be of minimum total weight as well. This problem is known under the name "Travelling Salesman Problem" and is examined in the following section.

10.3 Travelling salesman problem

In this chapter we have seen two different circuits with great practical importance in many fields; Euler and Hamilton circuits. In none of these were we interested in weighted graphs $G(V, E, w)$ where w is a function that assigns an arithmetic value on every edge. In weighted graphs, a new problem concerning circuits, with also great practical value arises. We would like to find the hamilton circuit with the minimum total weight. This problem is widely known under the name "Travelling salesman problem " (TSP) and is stated as follows:
A salesman wants to visit a number of cities which are connected. The distance between any two cities is known. Which is the order in which the salesman should visit the cities so that he minimizes the total distance we walks.

The problem is obviously to find the hamilton circuit with the minimum weight. Since finding a hamilton circuit is by itself NP-complete, it is obvious that the travelling salesman problem is NP-complete as well. If TSP was an easier to solve problem, we could solve this when asked for a Hamilton circuit!!! From this point, we will only consider complete graphs. This assumption makes the hamilton problem trivial but the TSP remains NP-complete.

10.3.1 Approximation algorithm

In many applications graphs represent points on the plane. Assigning the real distance between the two points as the weight of the respective edge is a common action. For these graphs the triangle inequality holds. This means that for any three vertices $u, v, z \in V$ we have that
$w(u, v) < w(u, z) + w(z, v)$

The problem remains NP-complete under this assumption, but at least there exist an approximation algorithm for this. An approximation algorithm sets a worst-case bound for the solution it gives. In this case we will prove that the algorithm we will discuss provides a solution with at most twice the cost of the optimal solution. The algorithm is based on a spanning tree of the graph.

2-APPROXIMATIONTSP(G(V,E,w))
1 Select a vertex $r \in V$ as the root vertex
2 grow a minimum spanning tree from r using Prim's algorithm
3 let L be the list of vertices in a pre-order walk of T
4 return the hamilton circuit that visit the vertices in the order of L

At this point it is important to remember that the pre-order tree walk ordering is obtained by recording every vertex once, when it is first visited. This algorithm always returns a circuit and runs in $O(E) = O(V^2)$ time, which is needed by Prim's algorithm.

What is left, is to prove that the returned circuit has at most twice the cost of the optimal circuit.
Proof.

Let H^* be the optimal circuit with cost $c(H^*)$. We want to show that the cost $c(H)$ of the returned circuit H is $c(H) \leqq 2 * c(H^*)$. Since the tree T computed is a minimum spanning tree we have that

$$c(T) \leqq c(H^*) \tag{10.7}$$

The full walk W of the tree from its root to all vertices and back to the root means that we traverse every edge exactly twice. This means that

$$c(W) = 2 * c(T) \tag{10.8}$$

If we combine this with equation 10.7 we get

$$c(W) < 2 * c(H^*) \tag{10.9}$$

If W was a circuit this would be enough. Unfortunately, W visits some vertices more than once. But due to the triangle inequality we can avoid visiting a vertex u for the second time in the following way: Let v be the vertex we were before visiting u for the second time and w be the vertex we leave u for. We want to avoid the path $v \to u \to w$. We can do this by going directly from v to w. The triangle inequality guarantees that the cost of the new tour is lower since:
$w(v, w) < w(v, u) + w(u, w)$

Furthermore by avoiding traversing a vertex for the second time, we result with the pre-order sequence of the vertices which is described by the algorithm.

So the tour we obtain has total cost lower than c(W). If we extend this with equation 10.9 we complete the proof.

For graphs with this property, the TS problem is known as Euclidean-TSP to remind the ancient mathematician who established the first rules for geometry. While we have an approximation algorithm for the Euclidean-TSP it has been proved that no c-approximation algorithm exists unless P=NP. This is the reason we have to turn our interest to heuristic algorithm. Algorithms that do not guarantee anything, but they appear to perform well in practise.

10.3.2 Heuristic algorithms

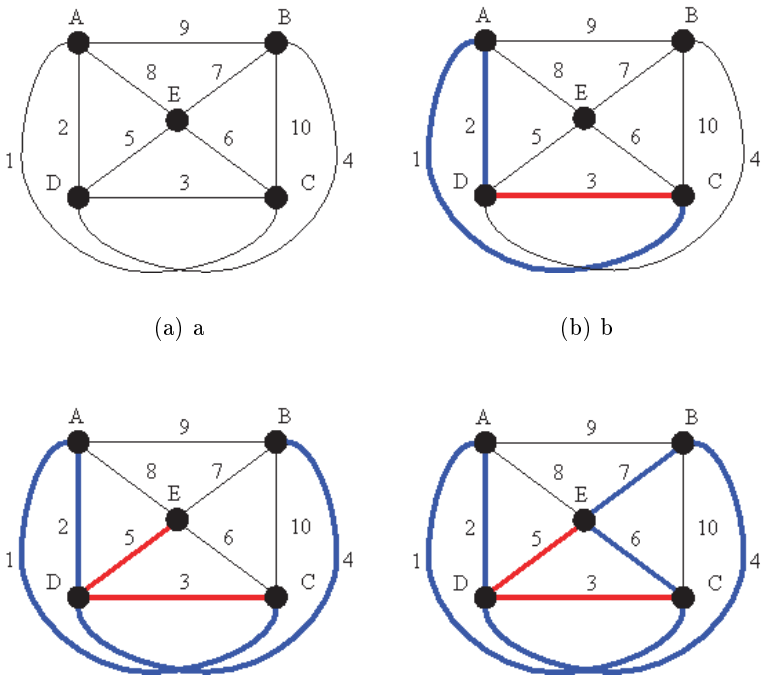
Even if the previous algorithm provides a solution for many interesting applications it is still not enough in the general case. It has been proved that in the general case there is no approximation algorithm for the travelling salesman problem, unless P=NP, which is highly unlikely. Here we will present some heuristic algorithms that can guarantee nothing but work satisfactory in practise.

Cheapest link algorithm

This is a greedy algorithm which works in a a way similar to Kruskal's algorithm for finding minimum spanning trees.

```
CHEAPEST LINK ALGORITHM CLA(G(V,E,w))
1  while  a hamilton circuit is not formed
2     do  Choose an edge with minimum weight randomly breaking ties
3         if  chosen edges do not form a smaller circuit and no vertex is incident to more than two edges
4           then
5                  add chosen edge to the path
6                  goto 1
```

Figure 10.11 shows the execution of "cheapest link" algorithm. 10.11(a) shows a graph (K5) on which we apply this algorithm: We choose AC as an edge of minimum weight. Then, since no restriction stops us we add AD. Now if we try to add DC we get a cycle ACDA (10.11(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(10.11(c)), so we do not use it. Instead we add EC and EB successively so we get a Hamiltonian cycle ADBECA (10.11(d)) with total cost 20



(a) a (b) b

(c) c (d) d

Figure 10.11: Various steps of the CL Algorithm

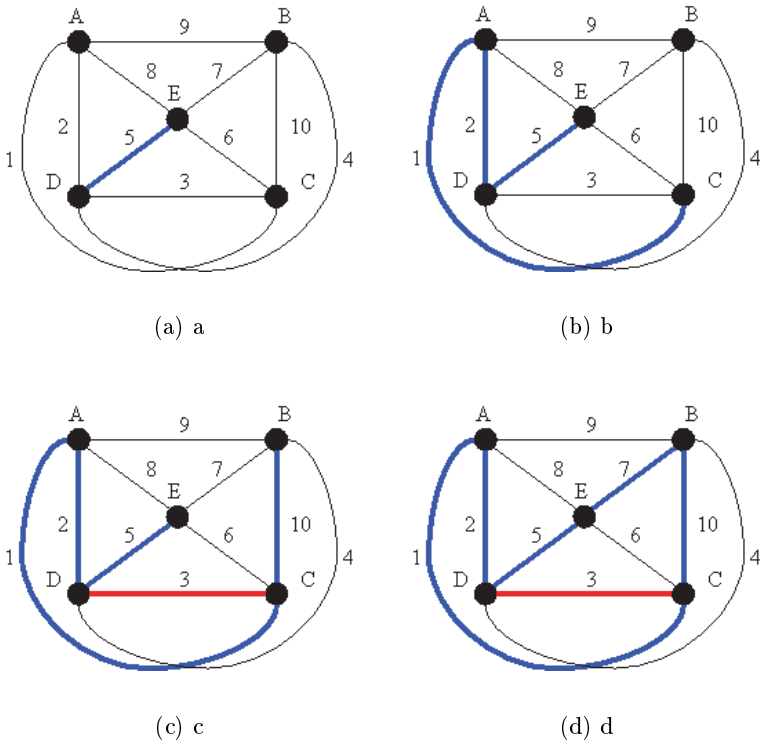
Nearest Neighbor Algorithm

This is a greedy algorithm which works in a a way similar to Prim's algorithm for finding minimum spanning trees. But since we want to grow a path and not a tree, the edge we select must be incident to the last visited vertex.

NEAREST NEIGHBOR ALGORITHM NNA(G(V,E,w))

```
1   Choose a vertex to start from
2   while not all vertices are visited
3       do Among the edge incident to the last visited vertex and any not yet visited vertex
4          chose the one with the minimum weight
5           visit the other endpoint of the edge
6
7   Travel back to the initial vertex
```

Figure 10.12 shows the execution of "cheapest link" algorithm. shows a graph (K5) on which we apply this algorithm: We choose E as the initial vertex. We choose the edge with the lower cost starting from E, ED10.12(a) . Similarly we traverse edges DA and AC (10.12(b)). Now we can not traverse the edge with the minimum cost CD since D has already been visited. Instead we we travel to B through CB(10.12(c)).Now all vertices are visited so we go back to the initial vertex through BE and obtain a Hamilton circuit EDACBE (10.12(d)) with total cost 25.



(a) a (b) b

(c) c (d) d

Figure 10.12: Various steps of the CL Algorithm

We observe that this algorithm depends on the choice of the first vertex. We canrun this algorithm $|V|$ times using a different initial vertex every time and keep the best solution.

This approach is called Repeated Nearest Neighbor Algorithm (RNNA) and is summarized as follows.

REPEATED NEAREST NEIGHBOR ALGORITHM RNNA(G(V,E,w))
1 **while** there are unmarked vertices
2 **do** Apply NNA selecting as initial an unmarked vertex v
3 Store the result and mark v
4 Among all circuits stored select the one with the minimum cost

The travelling salesman problem has great practical value. Designing the route in order to visit a number of places as quickly (economically, safely or anything that can be assigned as a weight for an edge) as possible is of great interest in many cases. From travelling to routing packets in a computer networks there are many fields where we would like to visit certain targets as efficiently as possible.