Algorithms in Graph Theory

tit.a.o.Prof. Dr. Herbert Fleischner

TU Wien, Algorithms and Complexity Group
fleischner@ac.tuwien.ac.at

March 11, 2016

This script is based on the lecture notes of "Algorithms in Graph Theory" held by tit.a.o.Prof. Dr. Herbert Fleischner at the TU Wien in the summer term 2012. It is written by

Projektass.(FWF) Dr. Martin Kronegger
TU Wien, Database and Artificial Intelligence Group
kronegger@dbai.tuwien.ac.at

A *graph* G is a finite set of vertices V together with a multiset of edges E (each connecting two not necessarily distinct vertices). We write $G = V \cup E$, unlike the usual way of writing $G = (V, E)$. The *size* of a graph $G = V \cup E$ is the number of edges denoted by $|E|$. The *order* of G is the number of vertices denoted by $|V|$. Figure 1 shows some examples of graphs.

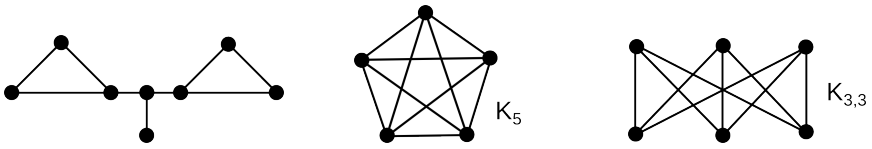
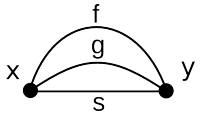


Figure 1: Example graphs.

A *loop* is an edge joining the same vertex, e. g. 

A *multiple edge* joins two vertices more than once, e. g. 

has the multiple edge $f = xy, g = xy, s = xy$. ($f = xy$ is read as "f is of the form xy" and not in terms of an algebraic equation.) Notice that $e = [x, y] = [y, x]$ is an older notation which can also be found in the literature. With $\lambda(xy)$ we denote the number of edges joining x and y. In the previous example, $\lambda(xy) = 3$.
A graph $G = V \cup E$ is called *simple* if $e \in E$ implies $\lambda(e) = 1$ and $e \neq xx$ for any $x \in V$ (does not contain a multiple edge and is loopless).
A simple graph K_n is called *complete* iff K_n has n vertices and for every two distinct vertices there is an edge joining them. A (simple) graph $G = V \cup E$ is called *bipartite* iff V can be divided into two disjoint sets s. t. there is no edge between two vertices of the same set. $K_{n,m}$ is called a *complete bipartite graph* iff V can be divided into two disjoint sets A, B with $|A| = n$ and $|B| = m$, s. t. $vw \in E$ for all $v \in A$ and $w \in B$, and there is no edge between any two vertices in the same set. See Figure 1 for K_5 and $K_{3,3}$.
For $e = xy \in E$ we say that e is *incident* to x, y respectively, or that x, y respectively, is incident to e. Vertices u and v are said to be *adjacent* if $uv \in E$. Edges e, f are said to be adjacent if there is $w \in V$ incident to e and f. E_v denotes the set of edges incident to v.
With the *degree* of $v \in V$ (abbreviated with $d(v)$) we denote the number of edges incident with v, counting loops twice. A vertex v is called *k-valent* if and only if $d(v) = k$. The *minimum degree* of a graph G is denoted with $\delta(G)$ and the *maximum degree* of G with $\Delta(G)$.

Lemma 1. *(Handshaking Lemma) Let $G = V \cup E$ be a graph. Then the following holds.*

$$\sum_{v \in V} d(v) = 2\,|E|$$

Corollary 2. *In a graph, the number of vertices of odd degree is even.*

A *directed graph* (*digraph*) D is a set of vertices V, together with a multiset A of *arcs* – denoted as $D = V \cup A$.

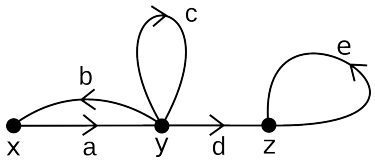
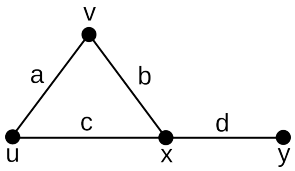
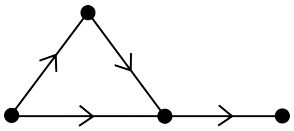


Figure 2: An example of a digraph.



(a) A simple graph G.



(b) An orientation of G of Figure 3a

Figure 3: Graphs, subgraphs and graph orientations.

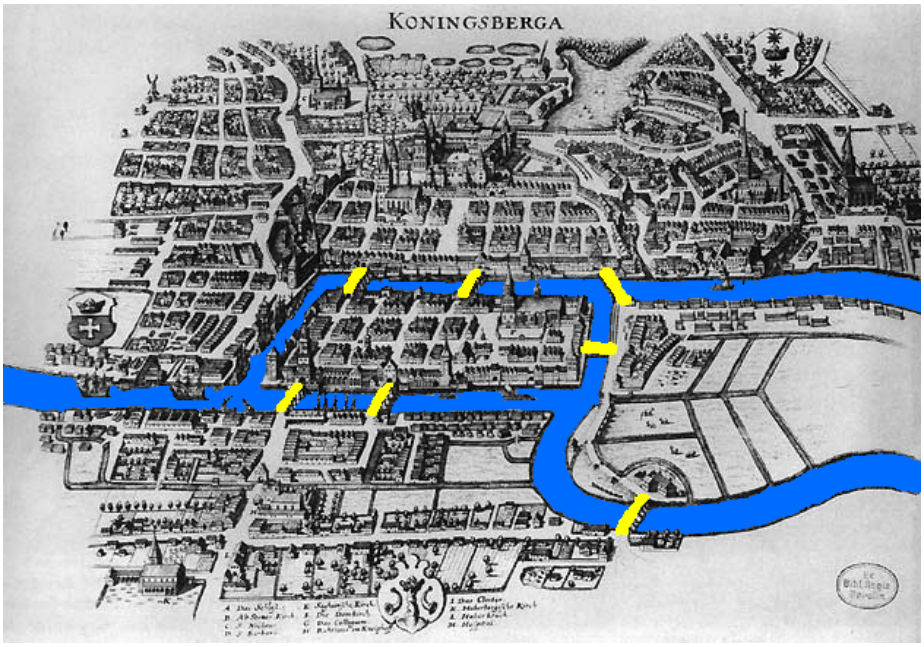
Arcs are denoted as ordered pairs of vertices, e. g. $a = (x, y)$, in which case we say that a is incident from x and incident to y; and x is said to be the *tail* of a and y is the *head* of a. Adjacency between two vertices, arcs respectively, are defined as in the case of graphs. A_v^+, A_v^- respectively, denotes the set of arcs incident from v, incident to v respectively. Then we write $A_v = A_v^+ \cup A_v^-$ to denote the set of arcs incident to v or incident from v.
The *in-degree* $id(v) = d^-(v)$ of a vertex $v \in V$ is the number of arcs that have v as their head. Analogously, the *out-degree* $od(v) = d^+(v)$, for any $v \in V$, is the number of arcs that have v as their tail.
For every digraph the following clearly holds.

$$\sum_{v \in V} od(v) = \sum_{v \in V} id(v) = |A|$$

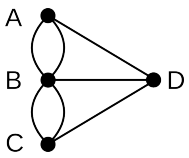
Figure 2 shows a digraph with arcs $a = (x, y)$, $b = (y, x)$, etc.
Given a graph $G = V \cup E$, $G' = V' \cup E' \subseteq G$. G' is a called a *subgraph* of G if $V' \subseteq V$, $E' \subseteq E$ and G' is a graph. For example, consider the graph G given in Figure 3a. Then $G' = \{u, x, y\} \cup \{c, d\}$ is a subgraph of G while $G'' = \{u, v, y\} \cup \{a, b, d\}$ is not a subgraph of G.
An *orientation* of an undirected graph $G = V \cup E$ is an assignment of exactly one direction to each edge, i. e., $D = V \cup \{(x, y) \text{ or } (y, x)\ \forall xy \in E\}$. Figure 3b shows an orientation of the graph given in Figure 3a.
Given a graph $G = V \cup E$. A *walk* $W(v_0, v_n)$ joining v_0 and v_n is defined as an alternating sequence of vertices and edges of G,

$$W = W(v_0, v_n) = v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n$$

s. t. $e_i = v_{i-1}v_i$, $1 \leq i \leq n$. The *length* of W, denoted as $l(W)$, is the number of edges in W. If in a walk $W(v_0, v_n)$ all edges are different, then this walk is called a *trail* joining v_0 and v_n. A walk is called *open* or *closed* depending on whether $v_0 \neq v_n$ or $v_0 = v_n$. A walk is called a *covering walk* if it contains all edges of G. If a covering trail is closed, then it is called an *eulerian trail*.
A graph $G = V \cup E$ is called *connected* if $\forall v, w \in V\ \exists W(v, w)$. Otherwise a graph is called *disconnected*. The empty graph is connected.



(a) A map of Königsberg as it was in Euler's days with highlighted bridges. (The picture is an edited version of the one from http://en.wikipedia.org/wiki/Königsberg.)

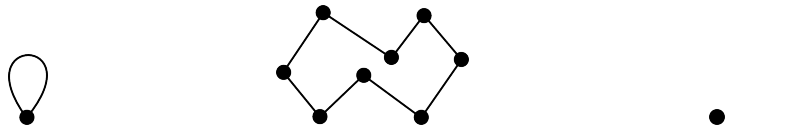


(b) The graph resulting from the "Seven Bridges of Königsberg" problem.

Figure 4: Seven Bridges of Königsberg.

A graph G is *eulerian* if $d(v)$ is even $\forall v \in V(G)$. (Remark: This does not mean that it is connected. Frequently, such disconnected graphs are often called "even graphs".) A digraph D is *eulerian* if $id(v) = od(v)$ $\forall v \in V(D)$.
Note: Already Euler in 1736 produced arguments starting with the problem of the "Seven Bridges of Königsberg" (now Kaliningrad, Russia) and explained why it is not possible to find a walk though the city that crosses each bridge exactly once. Figures 4a and 4b illustrate this problem.
An *open trail* is a *path* if no vertex is traversed more than once (so all vertices are different). In a path $P(v_0, v_n)$, the vertices $v_1, \ldots v_{n-1}$ are called the *internal* vertices of $P(v_0, v_n)$. A closed trail is a *cycle* (circuit) if all vertices are different except for $v_0 = v_n$ and if it contains at least one edge. A cycle of length k is also called *k-gon*. A cycle of length 2 is called *digon*; we define likewise *triangle*, *quadragles*, etc.

Proposition 3. *In an open walk $W(v_0, v_n)$ there is a subsequence $W'(v_0, v_n)$ which is a path.*



(a) Cycle. (b) Cycle. (c) Not a cycle.

Figure 5: Examples for a cycle, and a counterexample.

Proof. Let j_0 be the largest index s. t. $v_0 = v_{j_0}$. From $v_{j_0}, e_{j_0+1}, v_{j_0+1}$ we know that $v_{j_0+1} \neq v_{j_0} = v_0$. Let j_1 be the largest index s. t. $v_{j_1} = v_{j_0+1}$. By this we construct $v_{j_0}, e_{j_0+1}, v_{j_1}$. Continue this construction of a path as long as v_n has not been reached. If $v_{j_k} = v_n$, then stop. Clearly the constructed walk $W'(v_0, v_n) = v_{j_0}, e_{j_0+1}, \ldots, v_{j_k}$ is a path. □

Let $D = V \cup A$ be a digraph. Then

$$W(v_0, v_n) = v_0, a_1, v_1, a_2, v_2, \ldots, a_n$$

is a *walk* if $a_i = (v_{i-1}, v_i)$, i. e., the arcs are traversed in their direction. If however, $a_i = (v_{i-1}, v_i) \vee a_i = (v_i, v_{i-1})$ then W is called a *chain*. W is called a *simple chain*, if no vertex is repeated. In the Ford-Fulkerson-Algorithm below we will construct a special chain starting at the entry and ending at the exit of the network. However, in digraphs the concepts of trails, paths and cycles are defined as special cases of walk, just as in the case of graphs.
In a graph $G = V \cup E$, $v \in V$ is called an *isolated vertex* if $d(v) = 0$, whereas v is an *endvertex* if $d(v) = 1$. An edge $e \in E$ is called an *end-edge* if e is incident to an endvertex. An edge $e \in E$ is called a *bridge* if it is not contained in any cycle. Trivially, an end-edge is a bridge. In the graph in Figure 6, e and e' are bridges.
Note that eulerian graphs are bridgeless (why?). Analogous statement for digraphs holds as well.

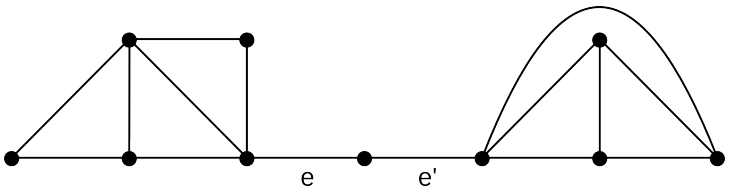


Figure 6: Graph with bridges e and e'.

A subgraph of G is called *component* of G if it is a maximal connected subgraph of G. (Note: A maximal connected subgraph cannot be enlarged by adding vertices/edges. A maximum connected subgraph is the largest possible connected subgraph, i. e., a maximum connected subgraph would be the largest possible component. However, a maximal connected subgraph needs not to be a maximum connected subgraph. See Figure 7 for an example.)

Proposition 4. *Let $G = V \cup E$ be a graph and R a relation. In the case that $\forall x, y \in V(xRy \Leftrightarrow \exists P(x, y))$ holds, then R is an equivalence relation. The equivalence classes of R are the vertex sets of the components of G.*

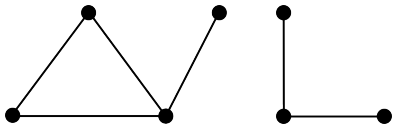
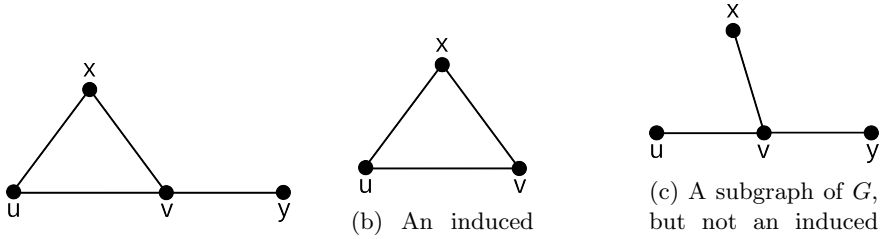


Figure 7: A disconnected graph whose smaller component is a maximal but not a maximum connected subgraph.

Proof. We show that R is an equivalence relation and that the equivalence classes are the vertex sets of the components of G and form a partition. The relation is reflexive, because every sequence starting at a vertex x and ending at the same vertex at the same time is a path. The relation is symmetric, because G is an undirected graph. For R being an equivalence relation it is left to show that R is transitive ($xRy, yRz \Rightarrow xRz$). The fact $\exists P(x,y) \wedge \exists P(y,z)$ gives us a walk $W(x,z)$ from x to z. By Proposition 3, we can construct a path $P(x,z)$ from x to z. Therefore R is an equivalence relation. R partitions $V = V_1 \dot\cup \ldots \dot\cup V_k$, $k \geq 1$. For $x \in V_i$, let G_i be the component defined by $x \in V(G_i)$. Clearly, $V(G_i) \subseteq V_i$ holds. Furthermore, $\forall x, y \in V_i \ \exists P(x,y) \subseteq G_i$ is obviously true. From this it follows that $V_i \subseteq V(G_i)$. Therefore, $V_i = V(G_i)$. □

Let $G = V \cup E$ be a graph. The subgraph of G *induced* by $V_0 \subseteq V$ is defined as $\langle V_0 \rangle = G_0 = V(G_0) \cup E(G_0) = V_0 \cup \{e = xy \in E \mid x, y \in V_0\}$. Analogously, the subgraph of G induced by $E_0 \subseteq E$ (*edge induced* subgraph) is defined as $\langle E_0 \rangle = G_0 = V(G_0) \cup E(G_0) = \{v \in V \mid v \text{ incident to some } e \in E_0\} \cup E_0$.



(a) A graph G.

(b) An induced subgraph of G.

(c) A subgraph of G, but not an induced subgraph.

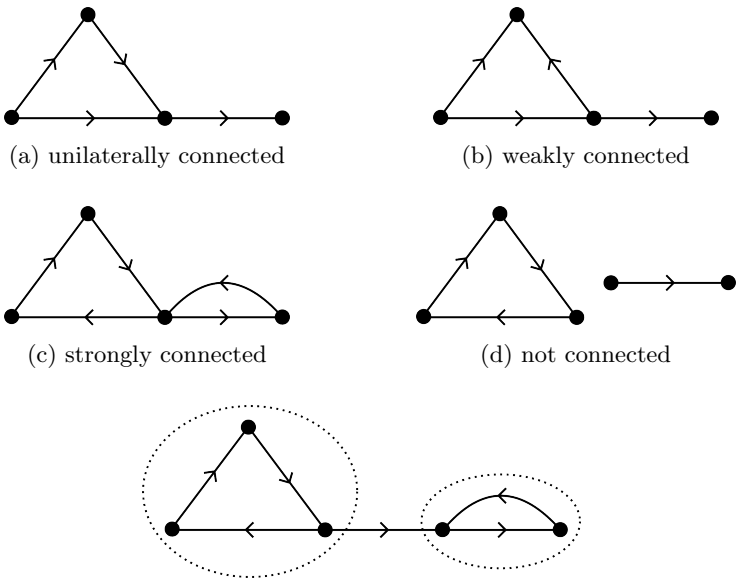
Figure 8: Example and counterexample for an induced subgraph.

Let $D = V \cup A$ be a digraph. D is called

- *weakly connected* if $G_D := V(D) \cup \{e = xy \mid (x,y) \vee (y,x) \in A\}$ is connected $\Leftrightarrow \forall x, y \in V \ \exists$ chain joining x and y
- *strongly connected* if $\forall x, y \in V \ \exists P(x,y) \wedge P(y,x)$
- *unilaterally connected* if $\forall x, y \in V \ \exists P(x,y) \vee P(y,x)$

Note that weak and strong connectedness induce an equivalence relation analogous to the case of graphs. Unilaterally connected digraphs do not induce an equivalence relation because the relation is not transitive, i. e., unilaterally connected components are not necessarily disjoint.
Consider the examples given in Figure 9 showing an unilaterally connected (9a), weakly connected (9b), strongly connected (9c) digraph and one which is not

connected (9d). Figure 9b is not unilaterally connected, because there is neither a path $P(u, v)$ nor a path $P(v, u)$. Figure 9e shows a digraph with two strongly connected components. The arc between these components does not belong to any strongly connected component (in contrast to the undirected case where every edge belongs to exactly one component).



(a) unilaterally connected

(b) weakly connected

(c) strongly connected

(d) not connected

(e) A graph with two strongly connected components.

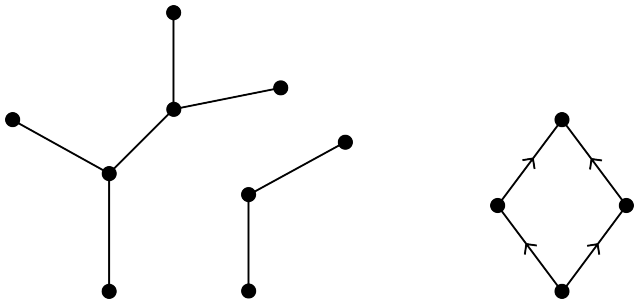
Figure 9: Different types of connectedness of digraphs.

A graph G or digraph D is called *acyclic* if it contains no cycle. An acyclic graph G is called a *forest*. A forest with precisely one component is called a *tree*. D is called a tree if G_D is a tree.
However, an acyclic weakly connected digraph need not be a tree (see Figure 10b), but it contains a vertex $v \in V(D)$ with $id(v) = 0$, which is called a *source* of D, and a vertex $w \in V(D)$ with $od(w) = 0$, which is called a *sink* of D.
A graph $G_1 \subseteq G$ is called a *spanning subgraph* of G if $V(G) = V(G_1)$. A *spanning tree* of G is a spanning subgraph of G which is a tree. Figure 10 shows an example of a forest and an acyclic digraph which is not a tree.

Theorem 5. *A graph $G = V \cup E$ is connected iff G contains a spanning tree.*

Proof. We proceed by showing both directions separately.

- "$\Leftarrow$": trivial.
- "$\Rightarrow$": Assume G is connected. If no edge belongs to a cycle of $G \Rightarrow G$ acyclic $\Rightarrow G$ is a tree. So, G is its only spanning tree. Suppose $\exists e \in E$ s. t. $\exists$ cycle C with $e = uv \in C$. Let $G_1 = G - e$. Consider an arbitrary path $P_G(x, y)$ in G with $x, y \in V$. If $e \notin P_G(x, y) \Rightarrow P_G(x, y)$ is a path $P_{G_1}(x, y)$ in G_1. Suppose $e \in P_G(x, y) \Rightarrow P_G(x, y) = x, \ldots, u, e, v, \ldots, y$, i. e., is of the form as illustrated in Figure 11.



(a) A forest. (b) Not a tree!

Figure 10: A forest and an acyclic digraph which is not a tree.

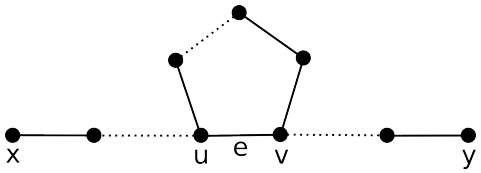
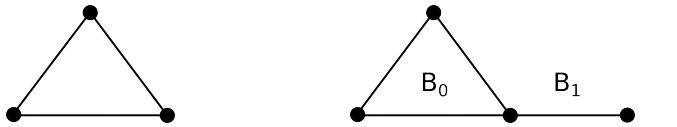


Figure 11: A path containing the edge e.

W.l.o.g. $C = v, e, u, P_C(u, v), v$. A walk W in G_1 is then defined as $W(x, y) = x, \ldots, u, P_C(u, v), v, \ldots, y$ where instead of going through e we use $P_C(u, v)$. Repeating this edge-elimination finally yields a graph G_k which is connected and acyclic $\Rightarrow G_k$ is a tree. As $V(G_k) = V(G)$, G_k is a spanning tree of G. □

A loopless graph $G = V \cup E$ is called *nonseparable* if G is connected and $G - v$ is connected $\forall v \in V$. A vertex $v \in V$ is called *cutvertex* if $G - v$ has more components than G. A maximal nonseparable subgraph of G is called a *block* of G. Figure 12 shows a nonseparable and a separable graph.



(a) nonseparable (b) not nonseparable = separable

Figure 12: A nonseparable and a separable graph.

Note: If a block B of G is a K_2 then the edge of B is a bridge of G.

Theorem 6. *Let $G = V \cup E$ be a loopless graph and B_1, B_2 be blocks of G. From $B_1 \neq B_2$ it follows that $B_1 \cap B_2 = \emptyset$ or $B_1 \cap B_2 = v \in V(G)$ and v is a cutvertex of G.*

Proof. Suppose B_1 and B_2 are different blocks with more than one vertex in common, i. e., $|V(B_1) \cap V(B_2)| \geq 2$. However, $V(B_1) - V(B_2) \neq \emptyset \neq V(B_2) - V(B_1)$ must hold; otherwise, $B_2 \subseteq B_1$ or $B_1 \subseteq B_2$ implying $B_1 = B_2$ in either case because of the maximality condition. Thus, there exists $z \in V(B_2) - V(B_1)$

adjacent to some $u \in V(B_1) \cap V(B_2)$. Since B_2 is nonseparable, there is a path

$$P_{B_2}(z, v) \subset B_2 - u \text{ s. t.} v \in V(B_1) \text{ but } (V(P_{B_2}(z, v)) - \{v\}) \cap V(B_1) = \emptyset.$$

In other words,

$$P_{B_2}(u, v) = u, uz, zP_{B_2}(z, v) \text{ satisfies } P_{B_2}(u, v) \cap B_1 = \{u, v\}.$$

Now it is straightforward to see that $B_1 \cup P_{B_2}(u, v) \supsetneqq B_1$ is nonseparable, contradicting the fact that B_1 is a block.
Suppose v is not a cutvertex of G. Let $x_i \in V(B_i)$ be adjacent to v for $i = 1, 2$. Then there exists $P(x_1, x_2) \subset G - v$. It follows for

$$P_i := P(x_1, x_2) \cup \{v, vx_i\}$$

that

$$B_1' = B_1 \cup P_2 \text{ and } B_2' = B_2 \cup P_1$$

are nonseparable graphs with $B_i \subsetneqq B_i'$ (see the preceding argument) contradicting B_i, $i = 1, 2$, being a block of G. □

The *block-cutpoint graph* $bc(G)$ of a graph G is defined in the following way.

- If B is a block of G, then $v_B \in V(bc(G))$;
- if c is a cutvertex, then $c \in V(bc(G))$,
  and there are no other vertices in $bc(G)$.
- $xy \in E(bc(G)) \Leftrightarrow x = v_B$ and y is a cutvertex of G contained in block B, or $y = v_B$ and x is a cutvertex of G contained in B.

Figure 13 shows an example of a block-cutpoint graph.

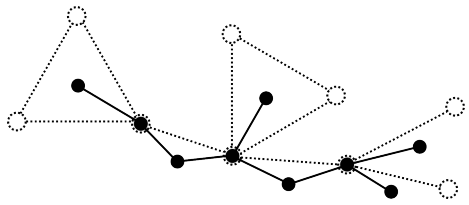
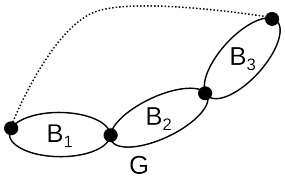


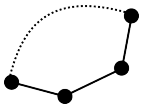
Figure 13: A graph G denoted with dotted vertices and edges and its block-cutpoint graph $bc(G)$ denoted with solid black vertices and edges.

Theorem 7. *Let G be a graph. Then $bc(G)$ is acyclic and #components of G = #components of $bc(G)$.*

Proof. The proof of Theorem 7 is left as an exercise. (*Hint:* Suppose $bc(G)$ has a cycle; then G has a subgraph as shown in Figure 14a. The cycle of $bc(G)$ is shown in 14b. However, the subgraph of G cannot be of the form as shown in Figure 14a, because it is nonseparable and would therefore not consist of more than one block.) □



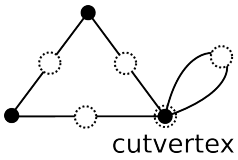
(a) A subgraph of G with several blocks.



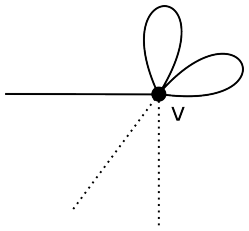
(b) The corresponding cycle of $bc(G)$.

Figure 14: Subgraphs of G and $bc(G)$ if $bc(G)$ would be cyclic.

A *subdivision* of a graph G results from subdividing edges of the form $xy \in E(G)$, i. e., remove xy, introduce a new vertex v_{xy} which is not present in G and add edges xv_{xy} and $v_{xy}y$. The *subdivision graph* $S(G)$ results from G by subdividing every edge of G by a new vertex. Clearly, a cutvertex remains a cutvertex when subdividing edges, only additional cutvertices may be introduced. However, when reversing this procedure a cutvertex should remain a cutvertex. Figure 15a illustrates the procedure of subdividing edges of a graph.



(a) A graph G with showing the result of the procedure of subdividing edges. The newly introduced vertices are dotted.



(b) Every loop is a block.

Figure 15: Subdivision of edges (left), and every loop is a block of the graph on the right.

To generalize the concept of nonseparable graphs to graphs with loops, the definition of a cutvertex can be extended. If $v \in V(G)$, $vv \in E(G)$, $d(v) \geq 3$ then v is also called a cutvertex. This means that each loop is considered as a block. For example, in Figure 15b every loop is considered to be a block. With these generalizations, $bc(G)$ can be defined as before.
A graph G is called a *block-chain* if $bc(G)$ is a path.

Theorem 8. *$G = V \cup E$ connected, $e \in E$ bridge $\Leftrightarrow V = V_1 \dot\cup V_2$ s. t. $\forall v_1 \in V_1, v_2 \in V_2$ every path $P(v_1, v_2)$ from v_1 to v_2 contains the edge e.*

Proof. We proceed by showing both directions separately.

- "$\Leftarrow$": Suppose $e = yx$ is not a bridge. Then $G - e$ is connected (see the proof of Theorem 5), and thus $P_{G-e}(v_1, v_2)$ exists for every $v_1 \in V_1, v_2 \in V_2$. This is a contradiction to our assumption that every path joining two vertices from V_1 and V_2 contains $e \Rightarrow e$ must be a bridge.
- "$\Rightarrow$": Assume $e = xy$ is a bridge. Then $G - e$ is disconnected; otherwise, $\exists P(x, y) \subset G - e$, i. e., $P(x, y), e$ is a cycle in G. Clearly $G - e$ cannot have more than two components. That is, $G - e$ has precisely two components

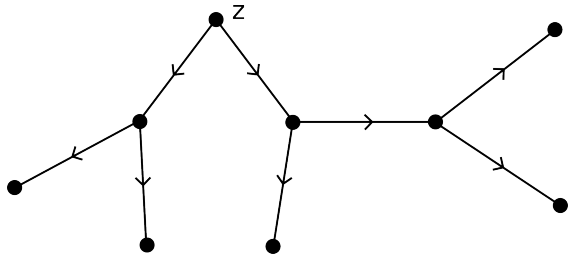
G_1 and G_2. W.l.o.g. assume $x \in \underbrace{V(G_1)}_{V_1}$, $y \in \underbrace{V(G_2)}_{V_2}$. Choose $v_1 \in V_1$, $v_2 \in V_2$. Then $\exists P_G(v_1, v_2)$. Let v_1' be the last vertex in $P_G(v_1, v_2)$ s. t. $v_1' \in V(G_1) = V_1$ and let v_2' be the first vertex in $P_G(v_1, v_2)$ s. t. $v_2' \in V(G_2) = V_2$, i. e., $P_G(v_1, v_2)$ is of the form $v_1, \ldots, \underbrace{v_1'}_{\in V_1}, f, \underbrace{v_2'}_{\in V_2}, \ldots, v_2$. (Note: $\nexists P_{G-e}(v_1, v_2)$). $\Rightarrow f = e$, since e is the only edge in G incident with a vertex in V_1 and V_2. □

Note: Deleting a bridge increases the number of components by exactly one. However, deleting a cutvertex may increase the number of components by an arbitrary number.

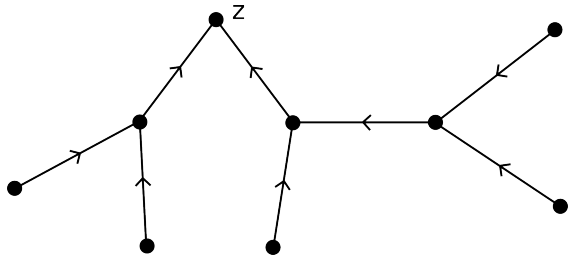
Theorem 9. *$G = V \cup E$ connected and loopless, $v \in V$ is a cutvertex $\Leftrightarrow$ $V - v = V_1 \dot\cup V_2$ s. t. $\forall v_1 \in V_1, v_2 \in V_2$ every path $P_G(v_1, v_2)$ contains v.*

Proof. Similar to the proof of Theorem 8; the proof details are thus left as an exercise. □

Let $D = V \cup A$ be an acyclic weakly connected digraph. If D is not a tree, then G_D contains a cycle C and $x, y \in V(C) \subseteq V$ s. t. $od_D(x) \geq 2$ and $id_D(y) \geq 2$ (see Figure 10b). Then D is an *out-tree* if $id(z) = 0$ for some $z \in V$ and $id(v) = 1, \forall v \in V - \{z\}$. z is the root of the out-tree. Figure 16a illustrates an out-tree. D is an *in-tree* if $od(z) = 0$ for some $z \in V$ and $od(v) = 1, \forall v \in V - \{z\}$. z is the root of the in-tree. Figure 16b illustrates an in-tree.



(a) Out-tree with root z.



(b) In-tree with root z.

Figure 16: An example for an out-tree and an in-tree with root z.

The digraph shown in Figure 17 has no spanning out-tree and no spanning in-tree (Exercise).

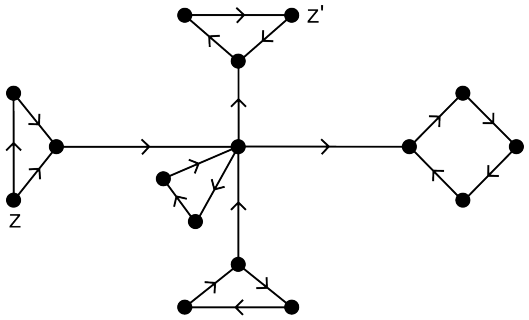


Figure 17: Digraph with no spanning out- or in-tree.

Theorem 10. *Digraph $D = V \cup A$ strongly connected, $z \in V$ arbitrarily $\Rightarrow \exists$ spanning in-tree with root z and a spanning out-tree with root z.*

Proof. Start with an arbitrary vertex z. Then $\exists P_D(z, v)$, $\forall v \in V$. Then $P_1(z, w) = T_1$ for some $w \in V$ is an out-tree with root z. Suppose $\exists x \in V - V(T_1)$ (otherwise we are finished). This means $\exists P_2(z, x)$. Let $u \in V(P_2(z, x))$ be the vertex s. t. $P_2(u, x) \subseteq P_2(z, x)$ has only u with T_1 in common. $\Rightarrow$ $T_2 = T_1 \cup P_2(u, x)$ is an out-tree with root z. Repeating this procedure we obtain a T_{k-1} which is an out-tree with root z, a path $P(u^{(k-1)}, y)$ where $y \notin V(T_{k-1})$ and $V(T_{k-1}) \cap V(P(u^{(k-1)}, y)) = u^{(k-1)}$, and $V - V(T_{k-1}) = V(P(u^{(k-1)}, y) - \{u^{(k-1)}\}$. Then $T_k = T_{k-1} \cup P(u^{(k-1)}, y)$ is an out-tree with root z and is spanning. A spanning in-tree with root z can be created analogously. □

Theorem 11. *Let G be a nonseparable graph, $|V(G)| \geq 2$, assume G is bridgeless (G is loopless, by the hypothesis.). Then $G - e$ is a block-chain for any $e \in E(G)$.*

Proof. Let $e = xy \in E(G)$ be an edge. Then $G - e$ is connected, $bc(G) = K_1$ and $bc(G - e)$ is a tree. Suppose $G - e$ has an endblock B s. t. $(V(B) - c) \cap \{x, y\} = \emptyset$, where c is a cutvertex. Then B is an endblock already in G, which is a contradiction. Thus $V(B - c) \cap \{x, y\} \neq \emptyset$ for every endblock of $G - e$. If $\{x, y\} \subseteq V(B) - c$, then $B \cup e$ is an endblock of G (which is also a contradiction). Therefore, every endblock B of $G - e$ contains precisely one of $x, y \neq c_B$, where c_B is a cutvertex. $\Rightarrow$ precisely two endblocks in $G - e$. $\Rightarrow$ $G - e$ is a block-chain. □

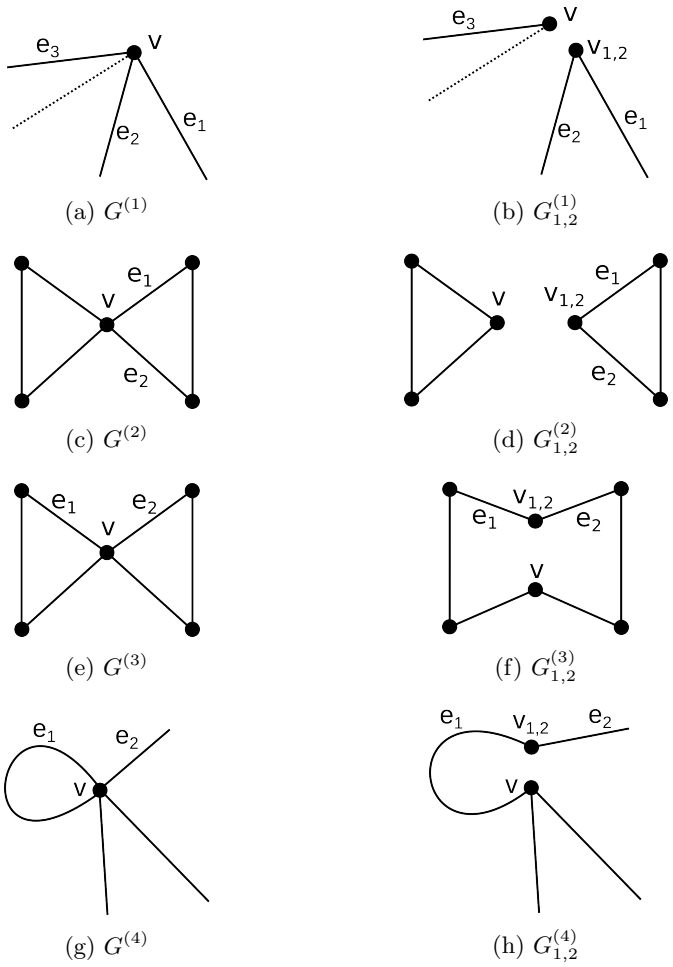
Let G be a graph. Suppose $\exists v \in V(G)$ with $d(v) \geq 3$. Let $E_v = \{e_1, e_2, e_3, \ldots\}$ be the edges incident with v. Then the graph $G_{1,2}$, which is obtained by *splitting* e_1, e_2 away from v, is defined in the following way.

$$V(G_{1,2}) = V(G) \cup \{v_{1,2}\} \text{ where } v_{1,2} \notin V(G).$$
$$E(G_{1,2}) = E(G) - \{e_1, e_2\} \cup \{e_1', e_2'\}, \text{ where } e_1 = x_1 v, e_2 = x_2 v \text{ and}$$
$$e_1', e_2' \text{ are incident with } v_{1,2} \text{ and } x_1, x_2 \text{ respectively.}$$

Theorem 12. *Let $G \neq K_2$ be nonseparable having a vertex v with $d(v) \geq 3$. Then $G_{1,2}$ is a block-chain, and $v, v_{1,2}$ are not cutvertices of $G_{1,2}$ and belong to different endblocks if $G_{1,2}$ is separable.*

Proof. The proof of Theorem 12 is similar to the proof of Theorem 11, and is therefore left as an exercise. □

Figure 18 shows on the left several graphs with a vertex v and on the right the result of splitting two edges away from v.



(a) $G^{(1)}$ (b) $G^{(1)}_{1,2}$

(c) $G^{(2)}$ (d) $G^{(2)}_{1,2}$

(e) $G^{(3)}$ (f) $G^{(3)}_{1,2}$

(g) $G^{(4)}$ (h) $G^{(4)}_{1,2}$

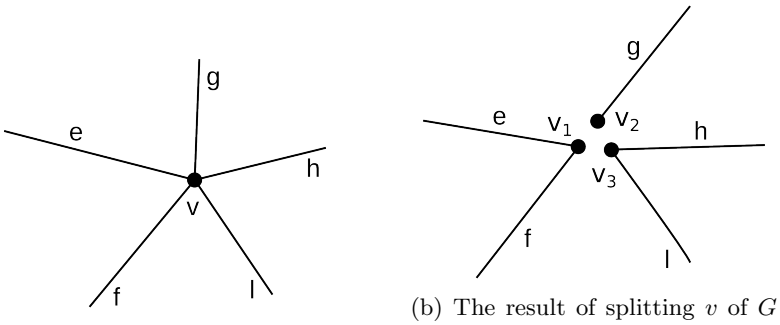
Figure 18: Examples of graphs on the left, and the result of splitting the edges e_1, e_2 away from vertex v on the right.

Splitting a vertex can also be generalized. Let $G = V \cup E$, $v \in V$ and $E_v = E_{v,1} \dot\cup \ldots \dot\cup E_{v,k}$. The graph G', resulting from splitting the vertex v w. r. t. $E_{v,1}, \ldots, E_{v,k}$, is then defined in the following way.

$$V(G') = \{V - v\} \dot\cup \{v_1, \ldots, v_k\} \text{ where } \{v_1, \ldots, v_k\} \cap V = \emptyset$$
$$E(G') = E(G) - E_v \cup \bigcup_{1 \le i \le k} \{xv_i \mid xv \in E_{v,i}\}$$

G' is called a *detachment* of G at v, also denoted as $G' = G_v$.

For example, consider the graph given in Figure 19a and $E_{v,1} = \{e, f\}, E_{v,2} = \{g\}, E_{v,3} = \{h, l\}$. The resulting graph after splitting v w. r. t. $E_{v,1}, E_{v,2}, E_{v,3}$ is shown in Figure 19b.



(a) A graph G.

(b) The result of splitting v of G w. r. t. to $E_{v,1}, E_{v,2}, E_{v,3}$.

Figure 19: Splitting v of G.

The splitting operation can also be repeated. Consider $V_0 \subseteq V, V_0 = \{w^{(1)}, \ldots, w^{(r)}\}$. Then $((\cdots((G_{w^{(1)}})_{w^{(2)}})\cdots)_{w^{(r)}})$ is a detachment of G at V_0.
A detachment at a vertex or at a set of vertices in a digraph is defined analogously.

Theorem 13. *G connected $\Leftrightarrow$ G has a detachment which is a tree.*

Proof. The proof of Theorem 13 is left as an exercise. (*Hint:* If G is connected and every edge is a bridge $\Rightarrow$ empty detachment. If an edge belongs to a cycle $\Rightarrow$ detach one end, and so on.) □

Theorem 14. *The following three statements are equivalent.*

1. *$G = V \cup E$ is a tree*
2. *G is acyclic and $|V| = |E| + 1$*
3. *G is connected and $|V| = |E| + 1$*

Proof. The proof of the Theorem 14 is left as an exercise. (*Hint:* Induction on the number of edges, starting from 0.) □

A *network* with integer capacities is defined in the following way. Let $D = V \cup A$ be a digraph with $V = \{\overline{a}, \overline{b}, v_1, \ldots, v_n\}$, $A = \{a_0 = (\overline{b}, \overline{a}), a_1, \ldots, a_m\}$ (a_0 is called *return arc*). In $A - a_0$ the following holds: there is no arc with $\overline{a}$ as head ($A_{\overline{a}} - a_0 = A^+_{\overline{a}}$) and there is no arc with $\overline{b}$ as tail ($A_{\overline{b}} - a_0 = A^-_{\overline{b}}$). $A^+_{\overline{a}}$ are the *entry arcs* of D and $A^-_{\overline{b}}$ are the *exit arcs* of D. Every $a \in A - a_0$ is assigned a *capacity* $c_a \in \mathbb{N} \cup \{0\}$ of a.
Let $D = V \cup A$ be a network, $a \in A \to \varphi_a \in \mathbb{R}$ be a function from the arcs to the real numbers and $A = \{a_0, \ldots, a_q\}$. Then the vector $\Phi = (\varphi_{a_0}, \ldots, \varphi_{a_q})$ is called a *flow* if

$$\sum_{a \in A^-_v} \varphi_a = \sum_{a \in A^+_v} \varphi_a, \forall v \in V$$

Φ is called a *feasible flow* if $0 \leq \varphi_a \leq c_a$ holds $\forall a \in A - a_0$. Clearly, the zero vector $\mathbf{0}$ is a feasible flow in any network. A feasible flow Φ in a network D with integer capacities is called a *maximum flow* if φ_{a_0} is a maximum.
Given a network D with integer capacities and a feasible integer flow Φ, the *Ford-Fulkerson Algorithm* shown in Algorithm 2 calculates a maximum flow. The algorithm repeatedly applies the marking procedure shown in Algorithm 1 as long as Φ can be enlarged.

Algorithm 1 The Marking Procedure used in the Ford-Fulkerson Algorithm.

```
1:  procedure MARKING-PROCEDURE(D, Φ)
2:      M := ā                          ▷ Comment: Initialize the set of marked vertices.
3:      repeat
4:          if v is marked and w is not marked, where a' = (v, w) ∈ A − a_0 then
5:              if φ_{a'} < c_{a'} then
6:                  M := M ∪ {w}
7:              end if
8:          end if
9:          if v is marked and w is not marked, where a' = (w, v) ∈ A − a_0 then
10:             if φ_{a'} > 0 then
11:                 M := M ∪ {w}
12:             end if
13:         end if
14:     until b̄ has been marked or no new vertex can be marked.
15:     return M
16: end procedure
```

Claim 1. *Let $\mathcal{M}$ be the set of marked vertices by the marking procedure shown in Algorithm 1. If $\overline{b} \in \mathcal{M}$, then Φ can be enlarged. If $\overline{b} \notin \mathcal{M}$, then Φ is a maximum flow.*

Proof. Let $D_m = \langle \{(u, v) \in A - a_0 \text{ s. t. } u, v \text{ marked}\} \rangle$. We show both statements separately.

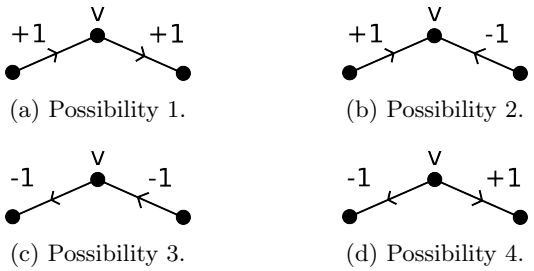
- Suppose $\overline{b} \in \mathcal{M} \Rightarrow \overline{b} \in D_m$. Let $D^{(0)} = \langle \overline{a} \rangle$. Clearly $D^{(0)}$ is weakly connected. The next marked vertex u is adjacent to $\overline{a}$. Therefore, let $D^{(1)} = \langle \{(\overline{a}, u\} \rangle$; $D^{(1)}$ is weakly connected. Suppose by induction that $D^{(k)}$ induced by the first k arcs involved by the marking procedure is weakly connected. Then the next step in the marking procedure involves a unique arc (x, y) (either $x \in D^{(k)}, y \notin D^{(k)}$ or $y \in D^{(k)}, x \notin D^{(k)}$). Then $D^{(k+1)} = D^{(k)} \cup \{(x, y), t\}$ where $t \notin D^{(k)}, t \in \{x, y\}$. Therefore, also $D^{(k+1)}$ is weakly connected. This means, if $\overline{b}$ has been marked, then D_m is weakly connected. So there is a simple chain $P(\overline{a}, \overline{b})$ in D_m with the following properties:
  - $P(\overline{a}, \overline{b}) = \overline{a}, (\overline{a}, u), \ldots, (y, \overline{b})$ and $\varphi_{(\overline{a},u)} < c_{(\overline{a},u)}$
  - $P(\overline{a}, \overline{b}) = \ldots, s, (s, t), t, \ldots$ and $\varphi_{(s,t)} < c_{(s,t)}$
  - $P(\overline{a}, \overline{b}) = \ldots, u, (w, u), w, \ldots$ and $\varphi_{(w,u)} > 0$

  Φ can now be increased to result in a feasible flow Φ' in the following way. $\forall (x, y) \in P(\overline{a}, \overline{b})$ where $x, (x, y), y \subseteq P(\overline{a}, \overline{b})$ set $\varphi'_{(x,y)} = \varphi_{(x,y)} + 1 \leq c_{(x,y)}$;

Algorithm 2 The Ford-Fulkerson Algorithm.

```
1: procedure FORD-FULKERSON(D)
2:     Φ = 0
3:     repeat
4:         M ← MARKING-PROCEDURE(D, Φ)
5:         if b̄ ∈ M then
6:             find a simple chain P(ā, b̄) s. t. v ∈ M, ∀v ∈ P(ā, b̄)
7:             for all (x, y) ∈ P(ā, b̄) do
8:                 if x, (x, y), y ⊆ P(ā, b̄) then
9:                     φ_(x,y) = φ_(x,y) + 1
10:                else                ▷ Comment: In this case y, (x, y), x ⊆ P(ā, b̄).
11:                    φ_(x,y) = φ_(x,y) − 1
12:                end if
13:            end for
14:            φ_a0 = φ_a0 + 1
15:        end if
16:    until b̄ ∉ M
17:    return Φ
18: end procedure
```

$\forall (y,x) \in P(\overline{a},\overline{b})$ where $x,(y,x),y \subseteq P(\overline{a},\overline{b})$ set $\varphi'_{(y,x)} = \varphi_{(y,x)} - 1 \geq 0$, for $\varphi'_{a_0} = \varphi_{a_0} + 1$ and $\forall a \in A - (\{a_o \cup P(\overline{a},\overline{b})\})$ set $\varphi'_a = \varphi_a$. When analyzing all four possibilities of changing the flow in a vertex (illustrated in Figure 20), it is clear that Φ' is still a feasible flow. In Figure 20a and 20c the sum of the incoming arcs is equal to the sum of the outgoing arcs (both sums are increased, respectively decreased, by one). Likewise, in Figure 20b and 20d, the sum of the incoming arcs and the sum of the outgoing arcs has not been altered.



(a) Possibility 1. (b) Possibility 2.

(c) Possibility 3. (d) Possibility 4.

Figure 20: The four possibilities of changing the flow in a vertex.

- Suppose $\overline{b} \notin \mathcal{M}$. Let $V_0 = V - \mathcal{M}$. Observe that $\overline{a} \notin V_0$ as $\overline{a}$ is always marked. Thus, V_0 and $\mathcal{M}$ are two disjoint sets with $\overline{a} \in \mathcal{M}$ and $\overline{b} \in V_0$. The general structure of the network is illustrated in Figure 21 (marked vertices are labeled with ⦿ and unmarked vertices with ⧈).

  Clearly, $\varphi_{a_i} = c_{a_i}$, for $1 \leq i \leq k$ and $\varphi_{b_j} = 0$, for $1 \leq j \leq l$; otherwise, a vertex in V_0 could be marked. Thus, $\varphi_{a_0} = \sum_{i=1}^{k} c_{a_i}$. □

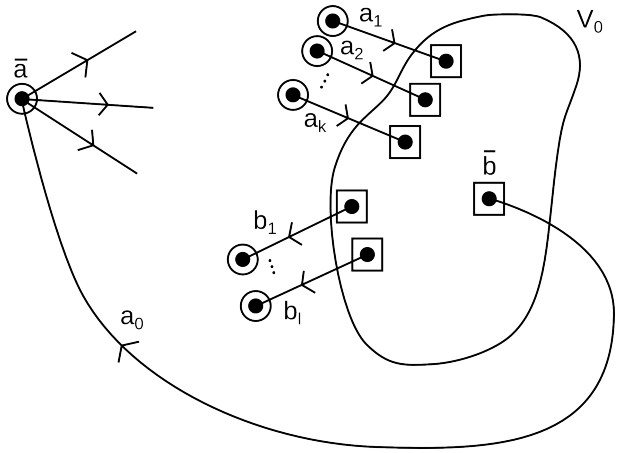


Figure 21: Graph illustrating the case where $b \notin \mathcal{M}$ in the proof of Claim 1.

Let D be a network, $V_0 \subset V(D) - \overline{a}$ and $\overline{b} \in V_0$. Then $A_0 = \{a = (x, y) \in A(D) \mid x \notin V_0, y \in V_0\}$ is called a *cut* of D. The *capacity of cut* A_0 is defined as $c_{A_0} := \sum_{a \in A_0} c_a$.
For every flow and every cut $\varphi_{a_0} \leq c_{A_0}$. Because $\varphi_{a_0} = \sum_{i=1}^{k} c_{a_i}$ for some special cut (see above) we also have $\varphi_{a_0} \geq \min_{A_0} c_{A_0}$. Thus we obtain the following result.

Theorem 15 (Max-Flow Min-Cut, Ford-Fulkerson's Theorem)**.** *The maximum flow in network D = minimum cut capacity over all cuts in D.*

Let $G = V \cup E$ be a graph and $x, y \in V$ be two vertices. Then the *local edge connectivity* $\lambda(x, y) =$ minimum #edges that need to be deleted in order to separate x and y $=$ min $|E_0|$ s. t. E_0 is an edge cut separating x and y. Furthermore, the *edge connectivity* of a loopless graph G is defined by $\lambda(G) = \min_{x,y \in V} \lambda(x, y)$. In the case of a bridgeless graph H having a loop, we set $\lambda(H) = 2$ (see Figure 24). For example, in the graph given in Figure 22a, at least two edges need to be deleted in order to make the graph disconnected while in the graph shown in Figure 22b at least 3 edges need to be deleted.

(a) $\lambda(G) = 2$ (b) $\lambda(G) = 3$

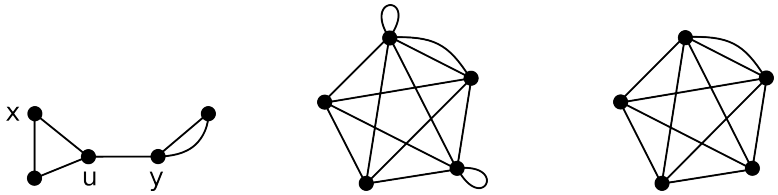
Figure 22: Examples of deleting edges to make a graph disconnected.

The *local connectivity* w. r. t. nonadjacent $x, y \in V$ is defined as $\kappa(x, y) = \min_{V_0 \subseteq V - \{x,y\}} \{|V_0| \mid x, y$ are in different components of $G - V_0\}$.
In Figure 23a, $\kappa(x, y) = 1$, because of $\{u\} = V_0$.

The *connectivity* of a loopless graph G is defined in the following way.

$$\kappa(G) = \begin{cases} \min\limits_{x,y \text{ non-adj.}} \kappa(x,y) & \text{if } \exists x,y \text{ s. t. } xy \notin E(G) \\ n-1 & \text{if } G \text{ contains a spanning } K_n \end{cases}$$

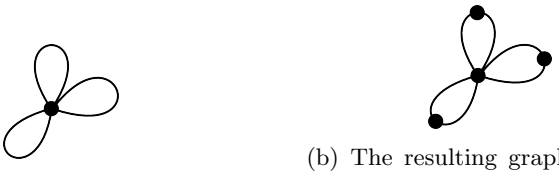
If a connected graph G has a loop, then we set $\kappa(G) = 1$.
Consider the graph given in Figure 23b which contains a spanning K_5: Since it has loops, $\kappa(G) = 1$. However, the graph G_0' obtained from G_0 by deleting the loops, satisfies $\kappa(G_0') = 4$ (see Figure 23c). Note that multiple edges do not affect the (local) connectivity, while they do affect the (local) edge connectivity.



(a) $\kappa(G) = 1$ (b) $\kappa(G_0) = 1$ (c) $\kappa(G_0') = 4$

Figure 23: Example graphs with their connectivity κ.

If G is loopless, then clearly $\lambda(G) \leq \delta(G)$ holds. In a graph with loops, one can think of subdividing the edges. For example let G be the graph shown in Figure 24a. Figure 24b shows the resulting graph after subdividing the edges. Then, clearly $\lambda(G_0) = 2$. In general, because of setting $\lambda(H) = 2$ for a bridgeless graph H having a loop, subdividing one or more edges of H leaves $\lambda(H)$ invariant.



(a) A graph G.

(b) The resulting graph G_0 after subdividing G of Figure 24a.

Figure 24: An example of connectivity in a graph with loops.

Let G be a graph and $E_0 \subseteq E(G)$ be an edge cut separating x and y, (illustrated in Figure 25); i. e., x and y lie in different components of $G - E_0$. Then $\rho_e(x,y)$ and $\rho(x,y)$ with $x, y \in V(G)$ are defined in the following way.

$$\begin{aligned} \rho_e(x,y) &= \text{maximum number of edge-disjoint paths joining } x \text{ and } y. \\ \rho(x,y) &= \text{maximum number of internally disjoint paths joining } x \text{ and } y. \end{aligned}$$

It is clear that $\rho_e(x,y) \leq |E_0|$. However, also the following theorem holds.

Theorem 16. *Let $G = V \cup E$ be a connected graph and $x, y \in V$ arbitrarily chosen with $x \neq y$. Then,*

$$\rho_e(x,y) = \min_{E_0 \subseteq E} \{|E_0| \mid E_0 \text{ separates } x \text{ and } y\}$$

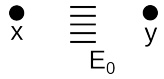
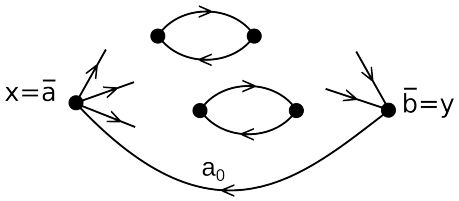
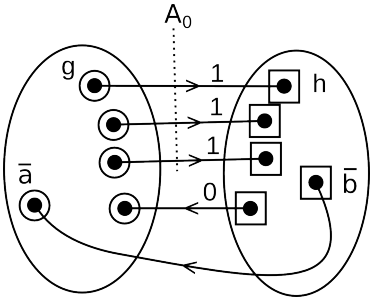


Figure 25: An illustration of an edge cut separating x and y where the lines indicate a set of edges.

Proof. Assume, w.l.o.g., that G is loopless (loops are never contained in a path). Construct a network D in the following way. $\overline{a} = x, \overline{b} = y$, $V(D) = V$, $(\overline{a}, u) \in A(D)$ if $xu \in E_x$, $(v, \overline{b}) \in A(D)$ if $vy \in E_y$, $a_0 = (\overline{b}, \overline{a}) \in A(D)$ and for all $st \in E$ with $\{s,t\} \cap \{x,y\} = \emptyset \Rightarrow (s,t) \in A(D)$ and $(t,s) \in A(D)$; i. e., st is replaced by two oppositely oriented arcs. Furthermore, for all $a \in A(D) - \{a_0\}$ let $c_a = 1$. The resulting network D with integer capacities is illustrated in Figure 26a.



(a) General structure of network D in the proof of Theorem 16.



(b) Marked vertices are labeled with ◉ and unmarked vertices with ▣.

Figure 26: The general structure of D and D^* with a maximum flow.

Then the maximum flow in D = minimum cut capacity c_{A_0} of a cut A_0. The network D with a maximum flow and the marking procedure associated with this flow is illustrated in Figure 26b. W.l.o.g. assume that $D' = (D - a_0) - \{a \in A(D) - a_0 \mid \varphi_a = 0\}$ is acyclic (otherwise, lowering the flow in the arcs of a cycle by 1, still yields a maximum flow). By construction, if $\{g,h\} \cap \{\overline{a}, \overline{b}\} = \emptyset$ and $(g,h) \in A(D)$ and g is a marked vertex, whereas h is unmarked, then $\varphi_{(g,h)} = 1$, and $(h,g) \in A(D)$ and $\varphi_{(h,g)} = 0$. Conversely, if $(t,s) \in A(D)$ with t unmarked, but s marked, then $\varphi_{(t,s)} = 0$, and $(s,t) \in A(D)$ with $\varphi_{(s,t)} = 1$. Likewise, if for arcs $(\overline{a}, v)$, $(w, \overline{b}) \in A(D)$ v is unmarked and w is marked then $\varphi_{(\overline{a},v)} = \varphi_{(w,\overline{b})} = 1$. Consequently, a cut A_0 in D corresponds to an edge-cut E_0 in G separating x and y.
It is now left to show that the maximum flow $= \varphi_{a_0} = |A_0| = |E_0| = \rho_e(x,y)$. The first three equations follow from the Ford-Fulkerson Theorem and the special type of network under consideration. To show the last equation we construct

a new digraph D'' from D' in the following way.

$$D'' = D' \cup \{a^{(i)} = (\overline{b}, \overline{a}) \mid i = 1, \ldots, \varphi_{a_0}\}$$

In D'', clearly $id_{D''}(w) = od_{D''}(w), \forall w \in V(D'')$ holds. In particular, we have $id_{D''}(x) = od_{D''}(x)$ and $id_{D''}(y) = od_{D''}(y)$ by construction of D''. That is, D'' is an eulerian digraph.

> Remark: A property of eulerian digraphs which we use here and will prove below without using networks is the following.
> D is an eulerian digraph with $A(D) \neq \emptyset \Leftrightarrow A(D) = C_1 \dot\cup \ldots \dot\cup C_k$ and $\langle C_i \rangle$ is a cycle for $1 \leq i \leq k$.

Thus, $A(D'') = C_1 \dot\cup \ldots \dot\cup C_k$. As D' has no cycles, each cycle of D'' contains some $a^{(i)}$. W.l.o.g. $a^{(i)} \in C_i, 1 \leq i \leq k = \varphi_{a_0}$. Clearly, $P^{(i)}_{D''}(\overline{a}, \overline{b}) := \langle C_i \rangle - \{a^{(i)}\}$ is a path in D'' which corresponds to a path $P^{(i)}_G(x, y)$ in G. All $P^{(i)}_{D''}(\overline{a}, \overline{b})$ are arc-disjoint, because all cycles $\langle C_i \rangle$ are arc-disjoint. Therefore, the paths $P^{(i)}_G(x, y)$ are edge-disjoint. This means, there are $k = \varphi_{a_0}$ edge-disjoint paths $P_G(x, y)$. Now, from $\rho_e(x, y) \geq \varphi_{a_0} = |E_0| \geq \rho_e(x, y)$ it follows that $\rho_e(x, y) = |E_0|$.

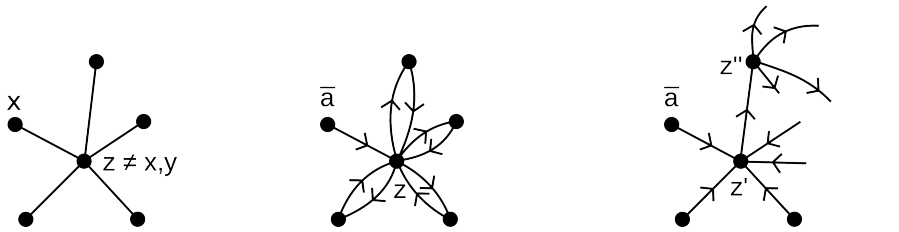
□

Theorem 17 (Menger)**.** *Let G be a graph, $x, y \in V(G)$ with $xy \notin E(G)$, then $\kappa(x, y) = \rho(x, y)$.*

Proof. Construct the network D as in the proof of Theorem 16. Construct a network D^* from D by forming a detachment at every $z \in V(D) - \{\overline{a}, \overline{b}\}$ as follows.

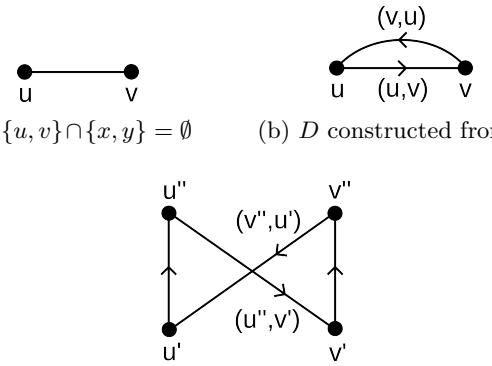
- z is replaced with $z', z'' \in V(D^*) - V(D)$
- $a^-_z \in A^-_z \subset A(D) \leftrightarrow a^-_{z'} \in A^-_{z'} \subset A(D^*)$,
  $a^+_z \in A^+_z \subset A(D) \leftrightarrow a^+_{z''} \in A^+_{z''} \subset A(D^*)$
- $(z', z'') \in A(D^*)$, called *pillar*
- $c_{(z', z'')} = 1$ and all other arcs in D^* have the same capacity 1 as the corresponding arc in D.

Figure 27 shows G locally and the construction of the corresponding networks D and D^*. Note that edge-disjoint paths in G do not necessarily correspond to arc-disjoint paths in D^* because there is exactly one arc leaving z'. However, disjoint paths from x to y in G correspond to disjoint paths in D^* joining $\overline{a}$ and $\overline{b}$ and thus also correspond to disjoint paths in D joining $\overline{a}$ and $\overline{b}$, and vice versa.
Figure 28 illustrates that an edge in G which is not incident to x nor to y, corresponds to a cycle of length 4 in D^*.
After calculating the maximum flow φ_{a_0} in D^* and applying the marking procedure w. r. t. this flow, the network D^* has the structure as shown in Figure 29 with $\varphi_{a_1} = \ldots = \varphi_{a_k} = 1$ and $\varphi_{b_1} = \ldots = \varphi_{b_l} = 0$. In all figures of the rest of this proof, the marked vertices are denoted with ◉, and the unmarked vertices with ▣.
Claim: $\rho_G(x, y) = \varphi_{a_0} = \kappa_G(x, y)$.



(a) G (b) D constructed from G. (c) D^* constructed from D.

Figure 27: An example for G and its constructed networks D and D^*.



(a) $\{u, v\} \cap \{x, y\} = \emptyset$ (b) D constructed from G.

(c) D^* constructed from D.

Figure 28: An edge in G (not incident to x nor to y) and the construction of the networks D and D^*.

In analogy to the proof of Theorem 16, we now obtain φ_{a_0} internally disjoint paths from $\overline{a}$ to $\overline{b}$ whose arcs have positive flow and thus yield equally many internally disjoint paths in D and in G.
It is easy to see that deleting in D^* one of the vertices of each of the φ_{a_0} arcs from the marked to the unmarked vertices, all of the aforementioned paths are destroyed. However, we want to show that in $D - \{\overline{a}, \overline{b}\}$ there are φ_{a_0} vertices whose deletion separates $D - a_0$ such that $\overline{a}$ and $\overline{b}$ belong to different weakly connected components of $D - a_0$. $\overline{a}$ remains marked and $\overline{b}$ remains unmarked.
Note that walking in D^* from $\overline{a}$ to $\overline{b}$ along one of the φ_{a_0} internally disjoint paths, call this path P, there is precisely one arc (q, r) where q is marked and r is unmarked, but none of the arcs of P has its tail unmarked but its head marked. We now mark the vertices of D starting from the final marking of D^* as follows. If the tail and the head of a pillar (u', u'') are both marked (unmarked, respectively), then we let $u \in V(D)$ be marked (unmarked, respectively). If for the pillar (u', u''), u' is marked, but u'' is unmarked then we let $u \in V(D)$ be marked. Note that by construction, u' cannot be unmarked whenever u'' is marked: otherwise $\varphi_{(u',u'')} = 0$ implying that all arcs incident to u' or incident from u'' have flow 0, and thus u'' could not be marked (see Figure 30 for an illustration).
Let $V_0 \subseteq V(D) = V(G)$ be defined as follows. If for $(s, t) \in A(D)$ the corresponding arc in D^* has flow 1, and if we have $t \neq \overline{b}$, and s is marked but t is

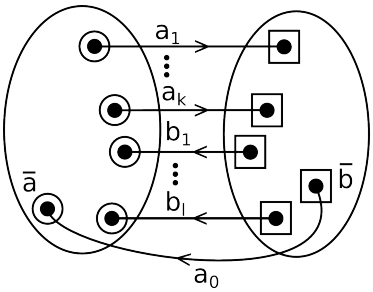


Figure 29: An illustration of D^* with the maximum flow.

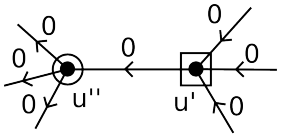


Figure 30: (u', u'') is a pillar where u' is unmarked and u'' is marked.

unmarked, then let $t \in V_0$. If, however, $t = \overline{b}$ and s is marked, then let $s \in V_0$ (note that $s \neq \overline{a}$ in this case, and $\overline{b}$ is unmarked in any case). We observe that among all arcs with $u \in V_0$ as head, there must be exactly one with flow 1 (the flow in D^* carries over to a flow in D). It now follows from the maximality of the flow in D^* and the corresponding marking in D, that

(i) V_0 corresponds to a set of $|V_0|$ vertices in D^* destroying all φ_{a_0} internally disjoint paths in D^* (as mentioned above), and

(ii) V_0 separates $D - a_0$ such that $\overline{a}$ and $\overline{b}$ belong to different weakly connected components of $(D - a_0) - V_0$ and thus separates x and y in G.

Thus we can say that for every arc of the cut of the maximum flow in D^*, we deleted one end in D. Thus, $|V_0| = \varphi_{a_0} =$# of internally disjoint paths from x to y in G. Clearly, $|V_0| \geq \kappa(x, y)$ and thus $\rho(x, y) \geq \varphi_{a_0} = |V_0| \geq \kappa(x, y)$. Furthermore, since every path from x to y in G must contain at least one vertex from any vertex cut separating x and y it follows that $\rho(x, y) \leq \kappa(x, y)$. Therefore, $\rho(x, y) = \kappa(x, y)$. □

Let G be a graph. The *line graph* $L(G)$ is defined as follows.

- $V(L(G)) = E(G)$
- $e, f \in V(L(G)), ef \in E(L(G)) \Leftrightarrow e, f \in E(G)$ are adjacent in G.

An example of a graph and its line graph is given in Figure 31.
Let $E_x = \{e_1, \ldots, e_{d(x)}\} \in E(G)$ be the set of edges incident with $x \in V(G)$, resp. $E_y = \{f_1, \ldots, f_{d(y)}\} \in E(G)$ the set of edges incident with $y \in V(G)$. Then L^* is defined as

$$L^* := L(G) \cup \{x, y\} \cup \{xe_i \mid 1 \leq i \leq d(x)\} \cup \{yf_j \mid 1 \leq j \leq d(y)\}.$$

In L^* the vertices x and y are nonadjacent. The proof of the following theorem is left as an exercise.

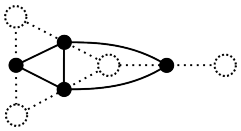


Figure 31: A graph G (dotted) and its line graph $L(G)$ (solid).

Theorem 18. $\kappa_{L^*}(x, y) = \lambda_G(x, y)$

Theorem 18 can be used to prove Theorem 16 with the help of Menger's Theorem. (*Hint:* Apply Menger's Theorem to L^* and modify the paths to correspond to paths in G. Then disjoint paths in L^* correspond to edge-disjoint paths in G).
Let G be a graph. G is called *n-connected* if $\kappa(G) \geq n$. From this definition it follows that, for example, a 3-connected graph is also 2-connected.

Theorem 19 (Whitney)**.** *G simple, n-connected $\Rightarrow \forall x, y \in V(G)$ $\exists n$ internally disjoint paths from x to y.*

Proof. Assume that $G = V \cup E$ is simple and n-connected. Therefore, $\kappa(G) \geq n$. Choose $x, y \in V$. We proceed by case distinction.

- x, y nonadjacent $\Rightarrow \kappa(x, y) \geq n \Rightarrow$ by Menger's Theorem $\rho(x, y) = \kappa(x, y) \geq n \Rightarrow \exists$ at least n internally disjoint paths joining x and y.
- $xy = e \in E$. Let $G' = G - e$. Obviously, $\kappa(G') \geq \kappa(G) - 1 \geq n - 1$. Apply Menger's Theorem to $G' \Rightarrow \exists$ at least $n-1$ internally disjoint paths joining x and y. However, the path x, e, y is internally disjoint from the others $\Rightarrow$ there are at least n internally disjoint paths joining x and y in G. □

Corollary 20. *Given an n-connected graph $G = V \cup E$ with $|V| \geq 2n$. Choose $V' = v_1, \ldots, v_n$, $W' = w_1, \ldots, w_n$ with $V', W' \subseteq V(G)$ and $V' \cap W' = \emptyset$. Then there are n totally disjoint paths joining vertices in V' to vertices in W'.*

Proof. W.l.o.g. assume that $v', w' \notin V(G)$. Form a graph $G^* = G \cup \{v', w'\} \cup \{v'v_i \mid 1 \leq i \leq n\} \cup \{w'w_j \mid 1 \leq j \leq n\}$. By this construction, $d(v') = d(w') = n$. Then $\kappa(G^*) \geq n$, because $\kappa(G) \geq n$ and $d(v') = d(w') = n$ (see also the construction of L^* preceding Theorem 18). Suppose that $\kappa(G^*) < n$. Then there is a vertex cut $S = \{x_1, \ldots, x_r\}$ with $r < n$, as depicted in Figure 32, that separates G^* i. e., $G^* - S$ is disconnected with v' and w' belonging to different components of $G^* - S$ (otherwise, S separates G which is impossible since $|S| < n$). However, $d(v') = d(w') = n > |S|$ implies that the components of $G^* - S$ containing x, y respectively, have at least two vertices each and thus S also separates G which is a contradiction to our assumption $\Rightarrow \kappa(G^*) \geq n$. By Menger's Theorem there are n internally disjoint paths $P_1(v', w'), \ldots, P_n(v', w') \subseteq G^*$. Then the paths $P_i = P_i(v', w') - \{v', w'\}$ are n totally disjoint paths in G joining vertices in V' to vertices in W'. □

Theorem 21 (Dirac)**.** *Let $G = V \cup E$ be an n-connected graph with $v_1, \ldots, v_n \in V$ chosen, $n \geq 2$. Then there exists a cycle containing $v_1, \ldots, v_n$. (Note that the order in which the vertices appear on the cycle cannot be prescribed.)*

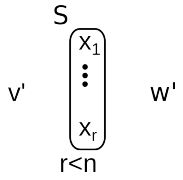


Figure 32: The vertex cut S in the proof of Corollary 20.

Proof. We proceed by contradiction. Assume there is no cycle containing all of $v_1, \ldots, v_n$. Let C be a cycle containing as many of the vertices $v_1, \ldots, v_n$ as possible. W.l.o.g. $v_1, \ldots, v_k$ are in $V(C)$, but $v_{k+1}, \ldots, v_n \notin V(C)$. By supposition, $k < n$.

- $l(C) = k$. Figure 33 illustrates this case.

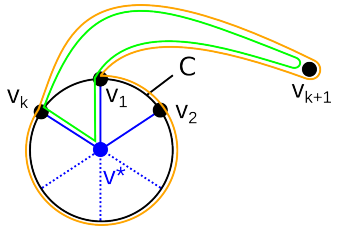


Figure 33: G^* in the proof of Theorem 21 in case of $l(C) = k$.

Form $G^* = G \cup \{v^*\} \cup \{v^* v_i \mid 1 \leq i \leq k\}$ (see Figure 33). Then $\kappa(G^*) = k$ (Delete in G^* the vertices $V(C)$; then v^* is isolated; and $G^* - v^* = G$ is n-connected.) $\Rightarrow \exists k$ internally disjoint paths from v_{k+1} to v^* (e.g., the green paths in Figure 33). Denote them as $P_1(v_{k+1}, v^*), \ldots, P_k(v_{k+1}, v^*)$. W.l.o.g. assume for $1 \leq i \leq k$ that the path $P_i(v_{k+1}, v^*)$ contains v_i. Let then $(C - \{v_1 v_k\}) \cup (P_1(v_{k+1}, v^*) - v^*) \cup (P_k(v_{k+1}, v^*) - v^*) =: C^* \supset \{v_1, \ldots, v_{k+1}\}$. Clearly $v^* \notin V(C^*) \Rightarrow C^*$ (Orange in Figure 33) is also a cycle in G. This is a contradiction to our assumption that C contains as many vertices of $\{v_1, \ldots, v_n\}$ as possible but not all of them.

- $l(C) > k$. Figure 34 illustrates this case.

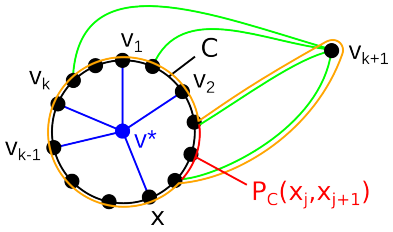


Figure 34: G^* in the proof of Theorem 21 in case of $l(C) > k$.

Let $x \in V(C) - \{v_1, \ldots, v_k\}$. Form analogously $G^* = G \cup \{v^*\} \cup \{v^* v_i \mid 1 \leq i \leq k\} \cup \{v^* x\}$. $\Rightarrow d_{G^*}(v^*) = k + 1$ and $k + 1 \leq n$. It follows analogously that $\kappa(G^*) = k + 1$. Again, there are internally disjoint paths

$P_1(v_{k+1}, v^*), \ldots, P_{k+1}(v_{k+1}, v^*)$. For $1 \leq i \leq k+1$, $P_i^-(v_{k+1}, x_i) \subset P_i(v_{k+1}, v^*)$ s. t. $v^* \notin P_i^-(v_{k+1}, x_i)$ and $P_i^-(v_{k+1}, x_i) \cap C = x_i$ (Denoted in green in Figure 34). W.l.o.g. assume the labels i are chosen s. t. $x_1, \ldots, x_{k+1}$ appear in this order on the cycle C. The cycle C can be written as $C = P_C(x_1, x_2) \cup P_C(x_2, x_3) \cup \ldots \cup P_C(x_{k+1}, x_1)$. Due to the pigeonhole principle $P_C(x_j, x_{j+1}) - \{x_j, x_{j+1}\} \cap \{v_1, \ldots, v_k\} = \emptyset$ for at least one j as there are $k+1$ paths but only k v_i's. Then $(C - P_C(x_j, x_{j+1})) \cup P_j^-(v_{k+1}, x_j) \cup P_{j+1}^-(v_{k+1}, x_{j+1})$ is a cycle in G that contains the vertices $v_1, \ldots, v_k, v_{k+1}$. This is a contradiction to our assumption that C contains as many vertices of $\{v_1, \ldots, v_n\}$ as possible. □

Corollary 22. *Let $G = V \cup E$ be a nonseparable graph with $|E| \geq 2$. Then, $e, f \in E \Rightarrow \exists$ cycle $C \supset e, f$.*

Proof. $|V(G)| \geq 2$ follows from the hypothesis, and the corollary is trivially true for $|V(G)| = 2$. Assume now that $|V| \geq 3$. $\kappa(G) \geq 2$ because G is nonseparable. Let $e = xy$, $f = uv$. Furthermore, let s_e resp. s_f be subdivision vertices of the edges e resp. f (illustrated in Figure 35).

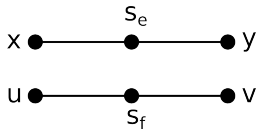


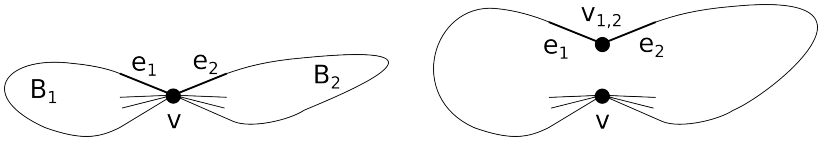
Figure 35: Subdivision vertices w. r. t. edges e and f.

$G' = (G - \{e, f\}) \cup \{s_e, s_f\} \cup \{s_e x, s_e y, s_f u, s_f v\}$. Clearly, $\kappa(G') \overset{(\geq)}{=} 2$. From Dirac's Theorem it follows that $\exists C' \ni s_e, s_f$ in $G' \Rightarrow C \supset e, f$ in G. □

Lemma 23 (Splitting-Lemma)**.** *G bridgeless, connected. $\exists v \in V$ s. t. $d(v) \geq 4$. Choose $e_1, e_2, e_3 \in E_v$ s. t. e_1, e_2 belong to different blocks if v is a cutvertex. $\Rightarrow G_{1,2}$ or $G_{1,3}$ is connected and bridgeless.*

Proof. We proceed by case distinction on whether v is a cutvertex or not.

A) Suppose v is a cutvertex. Let B_i be the block containing e_i with $i = 1, 2$. Let $H = B_1 \cup B_2$ (illustrated in Figure 36a). $H_{1,2}$ is shown in Figure 36b. We need to show that $H_{1,2}$ is nonseparable, which is true if B_1 or B_2 is a vertex with a loop. Assume, therefore, that $|V(H_{1,2})| \geq 4$. Then, $H_{1,2}$ is nonseparable $\Leftrightarrow \kappa(H_{1,2}) \geq 2 \Leftrightarrow x, y$ on a cycle for any $x, y \in V(H_{1,2})$.



(a) H (b) $H_{1,2}$

Figure 36: $H = B_1 \cup B_2$ and $H_{1,2}$ in the proof of Lemma 23 in the case where v is a cutvertex.

1) $x, y \in V(B_i) - \{v\}$ for $i \in \{1, 2\}$ i. e., both, x and y, are in the same block $\Rightarrow \exists$ cycle $C_i(x, y) \subset B_i$. If $e_i \notin E(C_i(x, y))$ then $C_i(x, y) \subset H_{1,2}$. Suppose $e_i \in E(C_i(x, y))$ (this case is illustrated in Figure 37). W.l.o.g. assume $i = 1$.

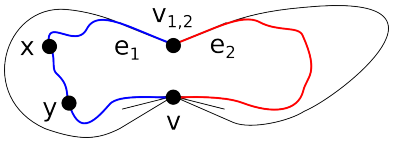


Figure 37: $x, y \in V(B_i) - \{v\}$

$C_1(x, y) \to P_1(v, v_{1,2}) \ni x, y$ (blue path in Figure 37). In B_2 $\exists C_2$ containing the edge $e_2 \to P_2(v_{1,2}, v)$ (red path in Figure 37). $P_1(v, v_{1,2}) \cup P_2(v_{1,2}, v)$ is a cycle in $H_{1,2}$ containing x and y.

2) $x \in B_1 - \{v\}$, $y \in B_2 - \{v\}$ (illustrated in Figure 38).

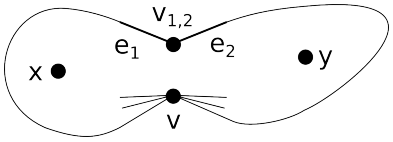


Figure 38: $x \in B_1 - \{v\}$, $y \in B_2 - \{v\}$

By Menger's Theorem (in Dirac form with e_1, e_2 subdivided) $\exists C_1(x) \subset B_1, C_1(x) \ni e_1, C_2(y) \subset B_2, C_2(y) \ni e_2$. $\Rightarrow P_1(v, v_{1,2}), P_2(v_{1,2}, v)$ is a cycle in $H_{1,2}$ containing both x and y. $C_1(x)$ corresponds to $P_1(v, v_{1,2}) \subset H_{1,2}$ and $C_2(y)$ corresponds to $P_2(v_{1,2}, v) \subset H_{1,2}$.

3) $y = v$, w.l.o.g. assume $x \in V(B_1)$ (illustrated in Figure 39).

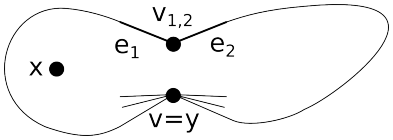


Figure 39: $y = v$, $x \in V(B_1)$

$\exists$ cycle $C_1(x) \subset B_1, e_1 \in C_1(x)$, which corresponds to a path $P_1(v, v_{1,2})$. In B_2 $\exists C_2(e_2) \ni e_2 \to P_2(v_{1,2}, v)$. Therefore, $P_1(v, v_{1,2}), P_2(v_{1,2}, v)$ is a cycle in $H_{1,2}$ containing x and $y = v$.

4) $v_{1,2}$ and v lie on a cycle in $H_{1,2}$ by analogous arguments.

In all cases $H_{1,2}$ is nonseparable (all blocks other than B_1 and B_2 are unaltered and therefore still blocks). From this it follows that $G_{1,2} = H_{1,2} \cup \bigcup_{i=3}^{r} B_i$ (where $B_1, \ldots, B_r$ are the blocks of G) is bridgeless since $H_{1,2}$ and $B_i, 1 \leq i \leq r$, are bridgeless. However, is $G_{1,2}$ connected? $P_G(x, y) \to P_{G_{1,2}}(x, y)$ if $E(P_G(x, y)) \cap \{e_1, e_2\} = \emptyset$ or $= \{e_1, e_2\}$. W.l.o.g. suppose $e_1 \in P_G(x, y)$ and $e_2 \notin P_G(x, y) \Rightarrow P_G(x, y) \ni e' \in E_v - \{e_1, e_2\}$ (see Figure 40 for an illustration).

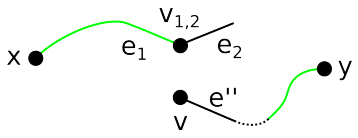


Figure 40: $P_G(x,y) \supset e' \in E_v - \{e_1, e_2\}$

B_2 is a block $\Rightarrow \exists P_{H_{1,2}}(v_{1,2}, v)$ s. t. $e_2, e'' \in P_{H_{1,2}}(v_{1,2}, v)$. Note that e_1 is not contained in this path, but possibly $e' = e''$.

$$e_1 \in P_{1,2}(x, v_{1,2}) \subset G_{1,2} \text{ corresponds to } P_G(x,v) \subset P_G(x,y);$$

$$e' \in P_{1,2}(v,y) \subset G_{1,2} \text{ corresponds to } P_G(v,y) \subset P_G(x,y)$$

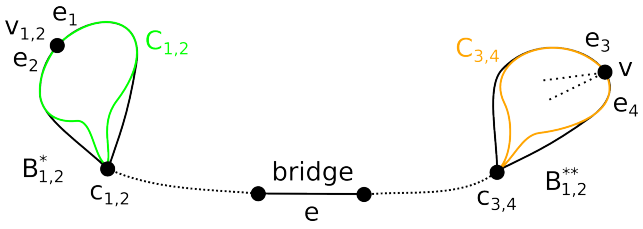
Therefore, we obtain a walk joining x and y in $G_{1,2}$

$$W_{1,2}(x,y) = P_{1,2}(x, v_{1,2}), P_{H_{1,2}}(v_{1,2}, v), P_{1,2}(v,y)$$

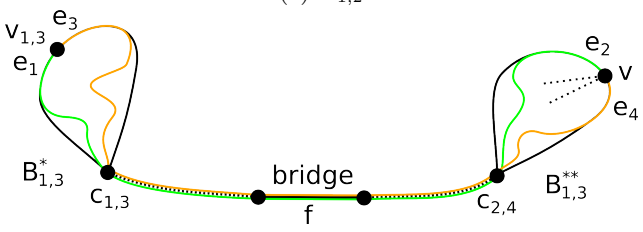
yielding in turn a path $P_{G_{1,2}}(x,y)$. Since x, y are chosen arbitrarily, we conclude that $G_{1,2}$ is connected and bridgeless.

B) Suppose v is not a cutvertex. Let B be the block of G containing v. It follows from Theorem 12 that $B_{1,2}$ is a block-chain in which both v and $v_{1,2}$ are not cutvertices and lie in different endblocks of $B_{1,2}$ if $B_{1,2}$ is a non-trivial block-chain. An analogous conclusion can be drawn w. r. t. $B_{1,3}$. Now, $B_{1,2}$ and $B_{1,3}$ are non-trivial block-chains but they could have bridges.

Suppose $B_{1,2}, B_{1,3}$ have bridges (note that we need to show only that one has no bridge). (see Figure 41).



(a) $B_{1,2}$



(b) $B_{1,3}$

Figure 41: $B_{1,2}$ and $B_{1,3}$

There is a cycle $C_{1,2}$ in $B^*_{1,2}$ containing e_1, e_2 (maybe containing also $c_{1,2}$), and a cycle $C_{3,4}$ in $B^{**}_{1,2}$ containing e_3, e_4. As there is the bridge

$e \in E(B_{1,2})$ it follows that $C_{1,2} \cap C_{3,4} = \emptyset$. $C_{1,2}$ corresponds to a path $P_{B_{1,3}}(v_{1,3}, v) \ni f$ and $C_{3,4}$ corresponds to a path $P'_{B_{1,3}}(v_{1,3}, v) \ni f$. $\Rightarrow$ $C_{1,2} \cap C_{3,4} \ni f$, which is a contradiction to $C_{1,2} \cap C_{3,4} = \emptyset$.

$\Rightarrow$ $B_{1,2}$ or $B_{1,3}$ is connected and bridgeless. W.l.o.g. $B_{1,2}$ is bridgeless; all other blocks in G are blocks in $G_{1,2}$. $\Rightarrow$ $G_{1,2}$ is bridgeless. However, $G_{1,2}$ is also connected because it results from G by replacing B with the connected graph $B_{1,2}$ in such a way that for every block $B^* \neq B$ in G we have $B^* \cap B = B^* \cap B_{1,2}$ in $G_{1,2}$. □

Suppose we have a 2-connected graph as shown in Figure 42.

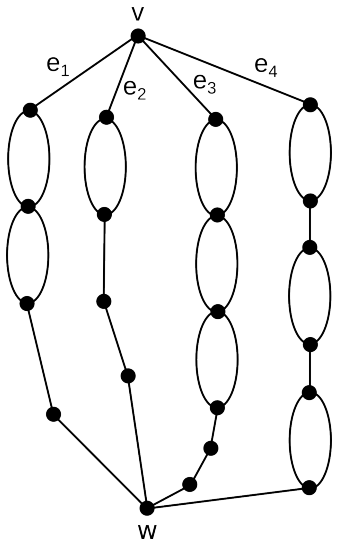


Figure 42: 2-connected graph.

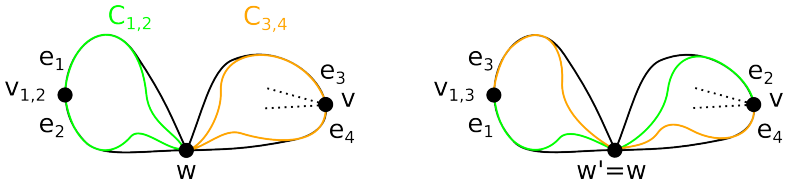
Corollary 24. *Let G be non-separable ($\neq K_2$) and v as before ($d(v) \geq 4$). Suppose $G_{1,2}$ and $G_{1,3}$ have a cutvertex ($e_1, e_2, e_3 \in E_v$). Then $G_{1,2}$ and $G_{1,3}$ have precisely one cutvertex w.*

Proof. Both $G_{1,2}, G_{1,3}$ are non-trivial block-chains with one cutvertex (see Figure 43). Suppose one of them, say $G_{1,2}$, has more than one cutvertex. The paths $P_{1,3}(e_1, e_2)$ and $P_{1,3}(e_3, e_4)$ would need to contain the cutvertices of $G_{1,3}$ (there is at least one cutvertex in $G_{1,3}$). On the other hand, the corresponding cycles in $G_{1,2}$ are disjoint (since $G_{1,2}$ has more than one cutvertex), which yields a contradiction. If the cutvertices w and w' in $G_{1,2}, G_{1,3}$ respectively, are different, the same argument would apply. (see Figure 43).

□

Theorem 25 (Characterization Theorem for eulerian graphs)**.** *Let G be a connected graph with $E(G) \neq \emptyset$. The following statements are equivalent.*

1. *G is eulerian.*
2. *G has an eulerian trail.*
3. *G has a cycle decomposition.*



(a) $G_{1,2}$ (b) $G_{1,3}$

Figure 43: $G_{1,2}$ and $G_{1,3}$ (both are non-trivial block-chains).

Proof.

- 1. $\rightarrow$ 2. Induction on the sum $\sum_{v \in V(G)}(d(v)-2) = \sigma(G)$. If $\sigma(G) = 0$ then G is a cycle C. Traversing C yields an eulerian trail of G. Suppose $\sigma(G) > 0 \Rightarrow \exists v \in V(G)$ s. t. $d(v) \geq 4$. Let $e_1, e_2, e_3 \in E_v$ s. t. e_1, e_2 are in different blocks if v is a cutvertex. By the Splitting-Lemma, $G_{1,2}$ or $G_{1,3}$ is connected and bridgeless. Note that $G_{1,2}$, $G_{1,3}$ are bridgeless because eulerian graphs are bridgeless.

  W.l.o.g. $G_{1,2}$ is connected. $\sigma(G_{1,2}) = \sigma(G) - 2$. Note that $v_{1,2}$ yields a zero-term. By induction, the graph $G_{1,2}$ has an eulerian trail $T_{1,2} = x, \ldots, e_1, v_{1,2}, e_2, \ldots, x \Rightarrow T_{1,2}$ corresponds to an eulerian trail T in G where $T = x, \ldots, e_1, v, e_2, \ldots, x$.

- 2. $\rightarrow$ 3. $T = x, \ldots, x$. Let T_0 be a shortest possible section of T starting and ending at some $y \Rightarrow T_0$ is a cycle $C(y)$. $T = x, \ldots, y, C(y), y, \ldots, x$. Then $T_1 = x, \ldots, y, \ldots, x$ with $E(C(y)) \cap T_1 = \emptyset$ is an eulerian trail in $G_1 = G - C(y)$ (delete all edges of $C(y)$ and all vertices that become isolated) if $E(G_1) \neq \emptyset$. $\Rightarrow$ if $G_1 = \emptyset$ then $\{C(y)\}$ is the only cycle decomposition of G. Otherwise, by induction $\exists$ cycle decomposition S_1 in G_1 and $S_1 \cup \{C(y)\}$ is a cycle decomposition of G.

- 3. $\rightarrow$ 1. $d_G(v) = \sum_{C_i \in S} d_{C_i}(v) \equiv 0 \bmod 2$ ($d_{C_i}(v) \in \{0, 2\}$), where S is a cycle decomposition of G. $\square$

Consider a connected eulerian graph G with $\delta(G) \geq 4 \Rightarrow \exists$ eulerian trail T, cycle decomposition S. T, S are called compatible if no section $e_i, v, e_{i+1} \in T$ belongs to the same element of S. With the help of the Splitting Lemma one easily shows that for a given cycle decomposition S there exists an eulerian trail T which is compatible with S. To find a cycle decomposition S compatible with a given eulerian trail T is an unsolved problem.

Theorem 26 (Characterization Theorem for eulerian digraphs)**.** *Let D be a weakly connected digraph ($A(D) \neq \emptyset$). The following statements are equivalent.*

1. *$\forall v \in V(D)$, $id(v) = od(v)$ (i. e., D is eulerian).*
2. *D has an eulerian trail.*
3. *D has a cycle decomposition.*

Proof. The proof is analogous to the proof for graphs and is therefore left as an exercise. $\square$

Theorem 27. *Let D be a weakly connected eulerian digraph and let T be an eulerian trail of D starting and ending at $z \in V(D)$. For every $v \in V(D) - \{z\}$ let a_v be the arc in T by which one leaves v for the last time ($a_v \in A_v^+$). Then*

$$B := \langle \{a_v \mid v \in V(D) - \{z\}\} \rangle$$

is a spanning in-tree of D with root z. Conversely, let B be a spanning in-tree of D with root z. Then there exists an eulerian trail T starting and ending in z s. t. for every $v \in V(D) - \{z\}$ the arc by which one leaves v in T for the last time, belongs to B.

Proof. $od_B(z) = 0$, $od_B(v) = 1$ for all $v \in V(D) - \{z\}$. Therefore, B spans D (note that the last arc of T is in B and incident to z). If B is acyclic and weakly connected, then B is a spanning in-tree with root z by definition.
Suppose B is not weakly connected. Then there exists a weakly connected component B' of B s. t. $z \notin V(B')$. Then $od_B(x) = 1$ for all $x \in V(B')$. This leads to a cycle $C \subset B'$ (see Figure 44); otherwise B' has a sink y and therefore $od_B(y) = 0$, which is a contradiction. $a_{i_1} = (v_{i_1}, v_{i_2})$ and $a_{i_2} = (v_{i_2}, v_{i_3})$ are such that a_{i_1} is the last arc of T by which one leaves v_{i_1} and then arrives at v_{i_2}; and likewise, a_{i_2} is the last arc of T by which one leaves v_{i_2}. Repeating this argument, we arrive at the ordering of indices

$$i_1 < i_2 < \ldots < i_k < i_1$$

where the arcs in the eulerian trail are a_1, a_2, etc. Thus we obtain $i_1 < i_1$, which is a contradiction. Thus, B is weakly connected, and one obtains the same contradiction if B contains a cycle. Thus, B is acyclic (and weakly connected). Therefore, B is a spanning in-tree with root z.

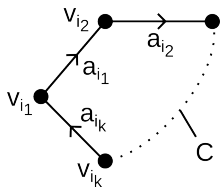


Figure 44: The vertices $v_{i_j} \in B'$ with $1 \leq j \leq k$ are not the vertices of the eulerian trail, but must be passed in this order.

Conversely, suppose a spanning in-tree rooted at z is given. Starting at z, traverse arcs of D arbitrarily subject to the condition that no arc is traversed more than once; and that at any $v \in V(D) - \{z\}$, the unique arc of B incident from V is traversed only if there is no other untraversed arc incident from v. Let the final trail be denoted by T. It follows from the construction of T that it is a closed trail; for, at any vertex $v \neq z$, at the last arrival at V by T there is still an arc by which one can leave v (since $id(v) = od(v)$). Suppose $A(D) \neq A(T)$. Then there is $x \in V(D) - \{z\}$ s. t. $A_x \not\subset A(T)$, but $A_y \subseteq A(T)$ for every $y \in V(P_B(x', z))$ where $P_B(x', z)$ is the unique path from x' to z in B and $(x, x') \in A(B)$. $A_{x'} \subseteq A(T)$ implies that $(x, x') \in A(T)$, implying that all other arcs of D incident from x are in T (by construction of T), which in turn implies that $A_x \subseteq A(T)$, a contradiction to the supposition. That is, $A(T) = A(D)$. The theorem now follows. □

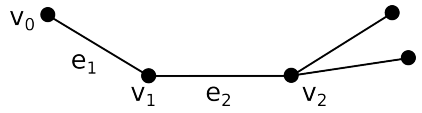
Algorithms for Constructing Eulerian Trails

Several parts of this chapter are taken directly from a book by Fleischner[1] where only the notation is adapted to the one used in this script.

Algorithm 3. *Splitting Algorithm*

Step 0 Given a connected, eulerian graph G of size $q > 0$, choose initial vertex $v_0 \in V(G)$, and let $T_0 = v_0$ be the initial trail. Set $H = G$ and $i = 0$.

Step 1 Suppose $T_i = v_0, e_1, v_1, \ldots, e_i, v_i$ was obtained by a (possibly empty) sequence of splitting away pairs of edges such that T_i appears as a path or cycle (in H) whose internal vertices are 2-valent.



If $i = 0$, choose $e_1 \in E_{v_0}$ arbitrarily and then go to **Step 1.2**. If $i \neq 0$, set $f_1 = e_i$.

Step 1.1 If $d_H(v_i) > 2$, then choose $f_2, f_3 \in E_{v_i} \cap (E(H) - E(T_i))$ and form $H_{1,j}$ for $j = 2, 3$ according to the Splitting Lemma. [However, one does not need the full strength of the Splitting Lemma since $H_{1,j}, j = 2, 3$, is bridgeless anyway and $H_{1,2}$ being disconnected implies that f_1, f_2 form an edge-cut unless $f_1 = f_2$ is a loop. So, if the block $B(f_1) \ni f_1$ satisfies $|E(B(f_1)) \cap E_{v_i}| \geq 3$, then f_2, f_3 can be chosen arbitrarily. So we do not need to look at cut vertices and do not need to look for edges in a different block]. If $H_{1,2}$ is disconnected, set $H = H_{1,3}$; otherwise set $H = H_{1,2}$

Step 1.2 If $d_H(v_i) = 2$, then H remains unchanged.

Step 1.3 Set $e_{i+1} = v_i v_{i+1}$ for the edge not in T_i but incident with v_i in H. Define $T_{i+1} = T_i, e_{i+1}, v_{i+1}$. Set $i = i + 1$.

Step 2 If $i \neq q$, go to **Step 1**; otherwise go to **Step 3**.

Step 3 The cycle T_q is an eulerian trail of G if viewed as an edge sequence.

Figure 45 shows that for G which is not eulerian, $G_{1,2}$ and $G_{1,3}$ can both be connected and contain bridge(s).
Let us consider the running time of the Splitting Algorithm. We mentioned already before that $H_{1,2}$ and $H_{1,3}$ are bridgeless in any case since these graphs are eulerian. And testing for connectedness can be done in linear time. Thus we only need to determine the number of splitting operations which need to be performed in the worst case. If $H_{1,2}$ is disconnected we continue working with $H_{1,3}$ where $d_{H_{1,3}}(v) = d_G(v) - 2$, and all other degrees are unchanged. Thus

[1] Herbert Fleischner, Eulerian Graphs and Related Topics, Annals of Discrete Mathematics, Vol. 50, ISBN: 978-0-444-89110-5

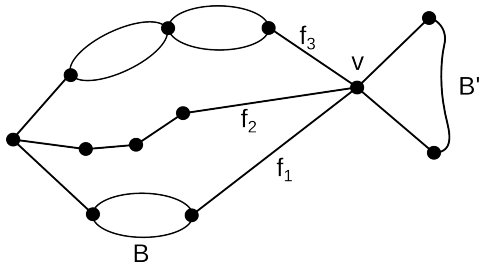


Figure 45: $G_{1,2}$ and $G_{1,3}$ can both be connected and contain bridge(s).

the number of splitting operations needed to ultimately reduce v to a 2-valent vertex, is at most

$$d_G(v) - 2$$

and the corresponding number of tests for connectedness is

$$\frac{d_G(v) - 2}{2}.$$

Clearly, the splitting operations as such can be performed in constant time, whereas the determination of a vertex of degree > 2 may be done in linear time. Altogether, the ultimate replacement of v by $\frac{d_G(v)}{2}$ 2-valent vertices in the corresponding connected eulerian graph requires at most

$$\left(\frac{d_G(v) - 2}{2}\right) \cdot P_1(G)$$

time units, where $P_1(G)$ is a linear polynomial in $|V(G)|$. That is, the running time to transform G into a cycle (which represents an eulerian trail of G), is at most

$$\begin{aligned}
\sum_{v \in V(G)} \left(\frac{d_G(v) - 2}{2}\right) P_1(G) &= P_1(G) \sum_{v \in V(G)} \left(\frac{d_G(v) - 2}{2}\right) \\
&= \frac{1}{2} P_1(G) \sum_{v \in V(G)} (d_G(v) - 2) \\
&= \frac{1}{2} P_1(G) \left(2\,|E(G)| - 2\,|V(G)|\right) \\
&= (q - p)\, P_1(G)
\end{aligned}$$

where $q = |E(G)|$ and $p = |V(G)|$. However, in the case of complete graphs G, the factor $q - p$ is

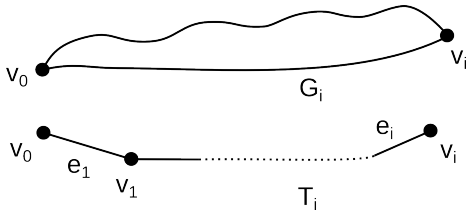
$$q - p = \binom{p}{2} - p = \frac{p(p-1)}{2} - p = \frac{p(p-3)}{2}.$$

That is, the total running time is bounded above by a cubic polynomial in $|V(G)|$.

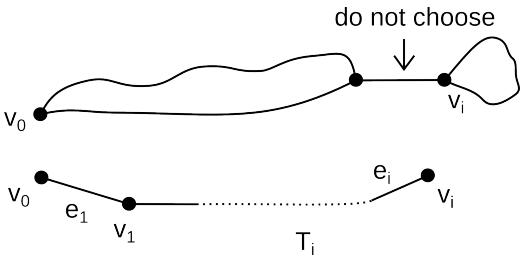
Algorithm 4. *Fleury's Algorithm*

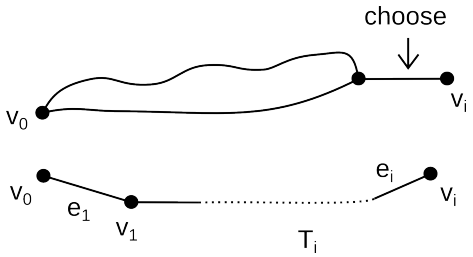
Step 0 G, v_0, T_0 as in **Step 0** of Algorithm 3 (Splitting Algorithm).

Step 1 Suppose the trail $T_i = v_0, e_1, \ldots, e_i, v_i$ has be chosen. Set $G_i = G - E(T_i)$.



Step 1.1 Choose $e_{i+1} \in E_{v_i} \cap E(G_i)$ such that e_{i+1} is not a bridge of G_i unless e_{i+1} is an end-edge of G_i (see Figures below).





Step 1.2 Set $T_{i+1} = T_i, e_{i+1}, v_{i+1}$ where $e_{i+1} \in E_{v_i} \cap E_{v_{i+1}}$.

Step 1.3 Set $i = i + 1$.

Continue with **Step 2** and **Step 3** as in the Splitting Algorithm.

We observe that the working of Fleury's Algorithm is justified by the working of the Splitting Algorithm. If we identify $v_0 \in V(G_i)$ with $v_0 \in V(T_i)$ and $v_i \in V(G_i)$ with $v_i \in V(T_i)$, then we are back in the construction of T_i in the Splitting Algorithm. This affinity between the Splitting Algorithm and Fleury's Algorithm shows that the running time of Fleury's Algorithm is bounded above by a cubic polynomial in $|V(G)|$.

Algorithm 5. *Hierholzer's Algorithm*

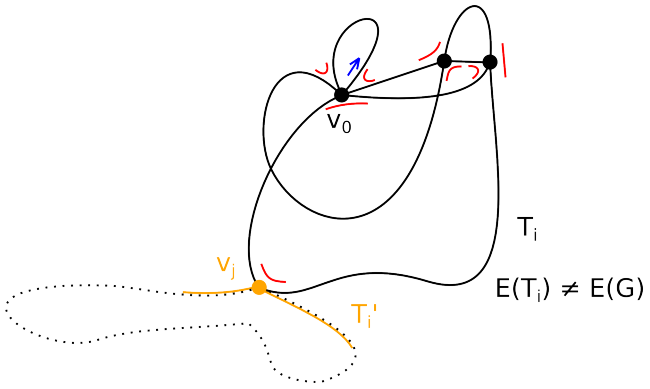
Step 0 The same as in Algorithms 3 and 4.

Step 1 Suppose $T_i = v_0, e_1, \ldots, e_i, v_i$ has been constructed. If $E_{v_i} - E(T_i) \neq \emptyset$, choose $e_{i+1} \in E_{v_i} - E(T_i)$ arbitrarily, set $T_{i+1} = T_i, e_{i+1}, v_{i+1}$ where $e_{i+1} \in E_{v_{i+1}}$, and set $i = i + 1$; otherwise, go to **Step 3**.

Step 2 Repeat **Step 1**.

Step 3 (note that all edges in E_{v_i} have been traversed by T_i). If $l(T_i) = q$ go to **Step 4**; otherwise, backtrack on T_i until v_j is reached with $E_{v_j} - E(T_i) \neq \emptyset$. Set $v_0' = v_j$ and produce T_i' in the component containing v_j in $G - E(T_i)$, analogously to T_i with respect to v_0 and T_0 by applying **Step 0** – **Step 2** until **Step 3** is reached again.

Now construct a larger trail T_i^* by inserting T_i' into T_i at v_j. That is, for $T_i = v_0, \ldots, e_j, v_j, e_{j+1}, \ldots, v_0$ we define $T_i^* = v_0, \ldots, e_j, T_i', e_{j+1}, \ldots, v_0$. Apply **Step 3** to T_i^*.



Step 4 The trail constructed is an eulerian trail of G.

Note that the backtracking procedure in T_i^* in Hierholzer's Algorithm need not start at v_0, but it suffices to start at v_0' by backtracking in T_i'.
In fact, Hierholzer's Algorithm works faster than both the Splitting Algorithm and Fleury's Algorithm. This is not immediately clear: the reason lies in the second part of **Step 2** of Algorithm 5 where no indication is given of how to find v_{i+1} for which $E_{v_{i+1}} - E(T_i) \neq \emptyset$. In fact, it all depends on how the graph has been stored and on the use of pointers. To see this, we need some additional considerations from which we derive a 'pidgin-algol' (pseudo-code) formulation of Hierholzer's Algorithm. To this end, let G be a simple connected eulerian graph (this assumption implies no real restriction since $S(S(G))$ is a simple graph for any graph G). The following is due to R.C. Read. Assume the graph to be stored as a number of adjacency lists – one for each vertex – each listing the adjacent vertices. Denote these lists by $L(x)$, $x \in V(G)$. The passing of an edge $e = uv$ from u to v by one of the subtrails T_i (see Algorithm 5) amounts to deleting in $L(u)$ the entry v and in $L(v)$ the entry u. For this

deletion procedure it is sufficient to overwrite the entry [v in $L(u)$, u in $L(v)$ respectively], and then decrease by one the length of the list, that is, decrease by one the degree of the vertex. As for the simultaneous deletion of u in $L(v)$, it is necessary to have a pointer to the u-entry in the v-list, and vice versa. With these data structures given we present R.C. Read's 'pidgin-algol' formulation of Hierholzer's Algorithm (see Algorithm 6).

Algorithm 6 R.C. Read's 'pidgin-algol' formulation of Hierholzer's Algorithm

```
 1: procedure HIERHOLZER(G)
 2:     u is any vertex of G.
 3:     HEAD and TAIL are stacks.
 4:     Initially HEAD = {u}, TAIL = ∅.
 5:     while HEAD ≠ ∅ do
 6:         while degree of top vertex, u, of stack HEAD is > 0 do
 7:             Let v be a vertex adjacent to u.
 8:             Add v to HEAD. (v becomes new u).
 9:             Delete edge uv from G.
10:             Decrease degrees of u and v by 1.
11:         end while
12:         while HEAD ≠ ∅ and top vertex u of HEAD has degree 0 do
13:             Remove u from HEAD.
14:             Add u to TAIL.
15:         end while
16:     end while
17:     Euler tour is TAIL.
18: end procedure
```

To see that the time and space complexity of this algorithm is $\mathcal{O}(q)$, observe first that the search for $v \in N(u)$ is easy – just take the first entry in $L(u)$. For, at any stage the first entry of $L(u)$ is a vertex v for which uv has not been passed yet. Moreover, the continuation of the subtrail T_i from v does not require any search procedure since v is on top of the stack HEAD (thus HEAD contains at most q objects at any given time), and $L(v)$ is immediately accessible. Second, if the construction of T_i is completed, that is, if T_i ends at its initial vertex u after having used all entries of $L(u)$, this vertex u is removed from HEAD as the top vertex of this stack and put on top of the stack TAIL. Then T_i is traced backwards by successively stacking the top vertex of HEAD on top of the stack TAIL until a top vertex w of HEAD is reached such that $d(w) > 0$. This is the only part of the algorithm which could absorb time unnecessarily. For, in this backtracking procedure one has to check at every top vertex x of HEAD whether $d(x) = 0$ or $d(x) > 0$, and this could cost time if one has to search for $d(x)$ first ($d(x) = |L(x)|$ since G is simple). However, given an appropriate storage of x and $L(x)$ respectively, one reaches $d(x)$ instantaneously so as to perform the comparison $d(x) = 0$ or $d(x) \neq 0$. At the above w for which $d(w) > 0$ the tracing of T_i' starts and ends. Then w is put on top of TAIL, and so forth. Thus, at any given time TAIL also contains at most q objects. Finally, an eulerian trail T has been read into TAIL in reverse order ('reverse' with respect to the orientation of the edges induced the first time an edge is used by the algorithm). Thus, reading T in TAIL from top to bottom this eulerian trail appears as a sequence in

accordance with the orientation given by passing the edges in the construction of the corresponding T_i.
Summarizing the preceding considerations on the working of Algorithm 6 we can say that every edge e is considered no more than three times: the first time in producing the corresponding subtrail T_i containing e; the second time in the backtracking procedure searching for w with $d(w) > 0$ (if such w exists); and finally in the reading of T in TAIL from top to bottom. Moreover, the latter part of **Step 3** of Algorithm 5 (Hierholzer's Algorithm), that is, combining T_i and T_i' to obtain T_i^*, does not require any particular effort inasmuch as this construction is implicitly contained in the backtracking procedure and in the way sections/segments of T are stacked in TAIL. In fact, this analysis shows that Algorithm 6 is linear.
Another algorithm for constructing eulerian trails has been developed by A. Tucker. It can be viewed as a combination of the Splitting Algorithm and Hierholzer's Algorithm. However, we need some additional concepts.
A *trail decomposition* $S = \{T_1, \ldots, T_k; k \geq 1\}$ is a set of pairwise edge-disjoint closed trails of G such that every $e \in E(G)$ belongs to some T_i, $1 \leq i \leq k$. Note that a graph is eulerian if and only if it has a trail decomposition.
A graph G is *k-regular* if $d(v) = k$ for every $v \in V(G)$.
Let T_1, T_2 be two edge-disjoint closed trails in a graph G with $v \in V(T_1) \cap V(T_2)$. Suppose

$$T_1 = x, \ldots, e_1, v, f_1, \ldots, x$$
$$T_2 = v, e_2, \ldots, f_2, v$$

where $x \in V(G)$, and for $i = 1, 2$, $e_i, f_i \in E(G)$.
Form $T^* = x, \ldots, e_1, v, e_2, \ldots, f_2, v, f_1, \ldots, x$ which is a closed trail with edge set $E(T_1) \cup E(T_2)$. We say that T^* results from a *κ-absorption* from T_1 and T_2.
The idea of Tucker's Algorithm is first to produce a trail decomposition $S = \{T_1, \ldots, T_k; k \geq 1\}$ for the connected eulerian graph G by pairing arbitrarily the elements of E_v for every $v \in V(G)$, and then to produce an eulerian trail by a sequence of κ-absorptions applied to pairs of connected subgraphs of G having a vertex in common and being induced by subsets of S.

Algorithm 7. *Tucker's Algorithm*

Step 0 Given the connected eulerian graph G, form a 2-regular detachment G_1 from G by splitting away adjacent pairs of edges as long as possible. Label the vertices of G_1 with the same symbols attached to the corresponding vertices in G. Set $i = 1$ and let c_i denote the number of components of G_i.

Step 1 If $c_i = 1$, set $T_i = G_i$ and go to **Step 3**. If $c_i \neq 1$, find two components T_i and T_i' of G_i such that $v_{i+1} \in V(T_i) \cap V(T_i')$ exists. Form a closed trail T_{i+1} by a κ-absorption at v_{i+1} applied to T_i and T_i'.

Step 2 Viewing T_{i+1} as a graph define $G_{i+1} = (G_i - (T_i \cup T_i')) \cup T_{i+1}$. Set $i = i + 1$, and go to **Step 1**.

Step 3 T_i is an eulerian trail of G.

It should be noted that once c_1 has been determined, the determination of c_i, $i > 1$, requires no efford. For, owing to **Step 1** and the definition of G_{i+1} in **Step 2** it follows that $c_{i+1} = c_i - 1$. Also, the construction of G_1 can be done instantaneously; and in the case $c_{i+1} \neq 1$ one can simply remain with T_{i+1} constructed in **Step 2**, and find T'_{i+1} such that v_{i+2} exists. However, a search for v_{i+2} can be avoided by using pointers again; they can be installed easily in the course of the construction of G_1. Thus, Algorithm 7 is also linear. It is left as an exercise to work out the details of this argument and to produce a more formalized version of Algorithm 7 analogous to the deduction of Algorithm 6 from Algorithm 5.

Mazes

The following historical remarks leading up to and including the formulation of the maze search problem and the corresponding algorithms are essentially taken from a book by Fleischner.[2]

The history of mazes and labyrinths, respectively, and the development of the first escape algorithms are at least as old as Greek mythology itself. For, as the story goes, Theseus used a thread given to him by Ariadne to track down the Minotaur in a maze and – after killing the creature – to find his way out again. Of course, mazes can be defined in different ways; e.g., as a system of catacombs (in "real life"), or (mathematically) as a set of unit squares in the euclidean plane, or as some other geometrical configuration. However, for our purposes, a *maze* is a graph G (digraph D, mixed graph H) which admits a covering walk W. Thus, G (D,H) must be (strongly) connected. In fact, the *maze search problem* can be formulated in the following way (unless stated otherwise, we restrict ourselves in the sequel to graphs):

> **Maze search problem (MSP).** Describe a general algorithm which constructs a closed covering walk W in a connected graph G such that, in the course of constructing W, this algorithm can only handle local information available at any vertex reached by W.

We note in passing that a labyrinth is in English often considered being different from a maze in that a labyrinth is seen as an intricate curve in the plane or 3-dimensional space, without intersecting itself, whereas a maze is – in the context of this course – simply a graph.

Observe that the oldest maze search algorithm (from a mathematical point of view) is Wiener's Algorithm: it is derived from Greek mythology, using Ariadne's thread. However, since Wiener's Algorithm is slower than Trémaux's and Tarry's Algorithms, we do not treat Wiener's Algorithm in this course. Note that the latter algorithms operate with local information only.

For the following considerations the expression $\lambda_W(e)$ denotes the number of uses of the edge e by a given walk W in the graph $G = V \cup E$. Thus, W is a covering walk if $\lambda_W(e) > 0$ for every $e \in E$, and it is called a *double tracing* if W is a closed covering walk with $\lambda_W(e) = 2$ for every $e \in E$. A double tracing W is called bidirectional if every edge of G is traversed in both directions.

[2]Herbert Fleischner, Eulerian Graphs and Related Topics, Annals of Discrete Mathematics, Vol. 50, ISBN: 978-0-444-89110-5

Proposition 28. *Every connected graph $G = V \cup E$ admits a bidirectional double tracing.*

Proof. For every $e = xy \in E$, replace e with the arcs (x, y) and (y, x) (if e is a loop, i. e., $x = y$, proceed analogously). The resulting digraph D satisfies $V(D) = V$, and it is eulerian ($id_D(v) = od_D(v)$ for every $v \in V(D)$). By Theorem 26, D has an eulerian trail T which – by construction of D from G – corresponds in G to a bidirectional double tracing. □

Proposition 29. *In a tree $G = V \cup E$, every closed covering walk W with $\lambda_W(e) \leq 2$ is a bidirectional double tracing.*

Proof. Let D_W be the digraph derived from G by replacing every $e \in E$ by $\lambda_W(e)$ arcs whose orientation is defined by the traversal of e in W. As a consequence, W corresponds to an eulerian trail T_W in D_W, i. e., D_W is eulerian. Considering an end-edge $e_0 = xy \in E$ with $d_G(y) = 1$ it follows that $(x, y), (y, x) \in A(D_W)$ since D_W is eulerian. Thus $D_1 := D_W - y$ is also eulerian with eulerian trail T_1 of D_1 corresponding to a closed covering walk W_1 in $G_1 = G - y$ with $\lambda_{W_1}(f) \leq 2$ for every $f \in E(G_1)$. Since the proposition is true for $|E| = 0, 1$, we conclude by induction that W_1 is a bidirectional double tracing and so is, by construction, W. □

We thus obtain the following corollary.

Corollary 30. *If G is a tree, then every double tracing of G is bidirectional.*

The next algorithm makes use of the following hypothesis.

> **Hypothesis.** Given a connected graph $G = V \cup E$. Whenever we reach a vertex $v \in V$ in the construction of the walk W, $\lambda_W(e)$ is known for every $e \in E_v$. Observe that the direction(s) in which e has been traversed by W so far, need not be known.

Algorithm 8. *Trémaux's Algorithm*

Step 0 Set $i = 0$, choose $v_0 \in V(G)$. Set $W = v_0$.

Step 1 Beginning at $v_i \in V(G)$ walk along any $e_i \in E_{v_i} - E(W)$ (this can be decided on the basis of the hypothesis). Set $W = W, e_i, v_{i+1}$ where $e_i = v_i v_{i+1}$. Set $i = i + 1$.

Step 2 Suppose $W = v_0, e_0, v_1, \ldots e_{i-1}, v_i$ has been constructed. If v_i is not an endvertex and $v_h \neq v_i$ for $0 \leq h < i$, go to **Step 1**. Otherwise, go to **Step 3**.

Step 3 (v_i is an endvertex or $v_i = v_j$ for some $j < i$). If $\lambda_W(e_{i-1}) > 1$ go to **Step 4**. If $\lambda_W(e_{i-1}) = 1$ set $e_i = e_{i-1}$, $v_{i+1} = v_{i-1}$ and set $W = W, e_i, v_{i+1}$. Set $i = i + 1$ and go to **Step 4**.

Step 4 If $\lambda_W(e) > 1$ for every $e \in E_{v_i}$, go to **Step 5**. Otherwise, choose $e \in E_{v_i}$ such that $\lambda_W(e)$ is minimal. Define $e_i = e$ and $v_{i+1} = y$, where $y = v_i$ if e is a loop, and $e \in E_{v_i} \cap E_y$ if e is not a loop. Set $W = W, e_i, v_{i+1}$, $i = i + 1$ and go to **Step 2**.

Step 5 W is a bidirectional double tracing.

Justification. We now justify Trémaux's Algorithm. First of all, **Step 3** and **4** of the algorithm guarantee that no edge is traversed more than twice. Also, the algorithm terminates in v_0 with every $e \in E_{v_0}$ traversed twice: for, upon arriving at any $v_i \neq v_0$, the algorithm has traversed edges once more towards v_i than from v_i, i. e., there must be an edge $f \in E_{v_i}$ which has been traversed only once so far. Thus, the final W is a closed walk, and it is easy to see that W traverses every edge in E_{v_0} once in each of the two directions.
Next we construct an auxiliary digraph D_W derived from the final W, as follows. For every edge $e = xy \in E(G)$, D_W contains an arc (x, y) if W traverses e from x to y. Consequently, if W traverses e twice, then D_W contains both arcs (x, y) and (y, x) (we leave it as an exercise to see that W does not traverse any edge twice in the same direction). It follows that W corresponds to an eulerian trail in D_W; i. e., D_W is an eulerian digraph.
Assume that the final W is not a bidirectional double tracing of G; i. e., some $f \in E(G)$ is traversed by the final W at most once.
Now delete in D_W all digons corresponding to a doubly traversed edge of G, to obtain a digraph $D^* \subseteq D_W$. Clearly, D^* is eulerian, so it has a cycle decomposition S^* whose elements contain arcs only which correspond to edges of G used precisely once by the final W.

a) If $S^* = \emptyset$, then $D^* = \emptyset$, i. e., W is a bidirectional double tracing of a proper subgraph of G. Consider a vertex w where $E_w - E(W) \neq \emptyset$. It follows that **Step 4** of the algorithm has been violated: For, upon leaving w along the edge $g = wy$ by which w has been reached for the first time, W didn't choose an edge e with $\lambda_{W'}(e) =$ minimum (W' is the subwalk of W ending at w just before traversing g for the second time). So, this case cannot happen.

b) $S^* \neq \emptyset$. Let $C^* \in S^*$,

$$C^* = v_{i_0}, a_{i_1}, v_{i_1}, a_{i_2}, v_{i_2}, \ldots, v_{i_0}$$

which corresponds in G to the cycle

$$C = v_{k_0}, e_{i_1}, v_{k_1}, e_{i_2}, v_{k_2}, \ldots, v_{k_0}$$

whose edges have been traversed precisely once by the final W and where the indices i_j and k_j, $j = 0, 1, \ldots$, refer to the position of these vertices and edges in the final W.

Looking at the sequence of indices derived from the edges of C

$$i_1, i_2, \ldots, i_r$$

we conclude that $i_1 < i_2$; otherwise (since v_{k_1} had been reached before, and the second part of **Step 3** would require to backtrack on e_{i_1} immediately). By the same token, we conclude $i_2 < i_3$, reaching ultimately the sequence of inequalities

$$i_1 < i_2 < \cdots < i_r.$$

However, we also obtain by the same argument the inequality

$$i_r < i_1;$$

i. e., $i_1 < i_1$ an obvious contradiction. Thus neither case a) or b) can happen, so the final W is a bidirectional double tracing of G.

□

The next algorithm also works with local information only and is based on the following hypothesis.

Hypothesis. At any vertex $v \in V(G)$ reached in the course of the construction of the walk W, the set $E^o_{v,W} \subseteq E_v$ of edges already passed from v is known. Moreover, the edge $e_{\text{in}}(v)$ by which $v \neq v_0$ has been reached for the first time is known. For the initial vertex v_0 of W we set $\{e_{\text{in}}(v_0)\} = \emptyset$.

Algorithm 9. *Tarry's Algorithm*

Step 0 Set $i = 0$, choose $v_0 \in V(G)$. Set $W = v_0$.

Step 1 Beginning at $v_i \in V(G)$ walk along an arbitrary $e_i \in E_{v_i}$ which has not yet been traversed from v_i nor by which v_i has been reached for the first time. Set $W = W, e_i, v_{i+1}$ $(e_i \in E_{v_{i+1}})$. Set $i = i + 1$.

Step 2 Suppose $W = v_0, e_0, v_1, \ldots e_{i-1}, v_i$ has been constructed. If $E^o_{v_i,W} \cup \{e_{\text{in}(v_i)}\} \neq E_{v_i}$, go to **Step 1**. Otherwise, go to **Step 3**.

Step 3 If $\{e_{\text{in}}(v_i)\} \subseteq E^o_{v_i,W}$ go to **Step 4**. Otherwise, set $e_i = e_{\text{in}}(v_i)$, $W = W, e_i, v_{i+1}$ $(e_i \in E_{v_{i+1}})$, $i = i + 1$; go to **Step 2**.

Step 4 W is a bidirectional double tracing in G.

Justification. Suppose the final walk W ends in v. If $v \neq v_0$ then the number of arrivals of W in v is by one larger than the number of departures of W from v. That is, by **Steps 1**, **2**, **3**, $E^o_{v,W} \neq E_v$, and thus W could be continued by **Steps 2** and **3**. Thus $v = v_0$, and so W is a closed walk using every $e \in E_{v_0}$ once in each direction (note that by **Steps 2** and **3**, W traverses no edge more than twice). Suppose W is not a bidirectional double tracing of G. Thus some edge of G is traversed at most once by W (**Steps 2** and **3** guarantee that no edge is traversed twice in the same direction).
Next construct D_W just as in the justification of Trémaux's algorithm. Whence D_W is an eulerian digraph with W corresponding to an eulerian trail T_W of D_W. Traversing T_W starting at v_0, let v_i be the first vertex at which T_W arrives for the first time and where W does not traverse every edge $e \in E_{v_i}$ twice. So for every $j < i$, W traverses every edge incident to v_j once in each direction. For $j = i - 1$, we have that T_W traversed (v_{i-1}, v_i) to reach v_i for the first time, and (v_i, v_{i-1}) is also in D_W because T_W traverses in v_{i-1} every $e \in E_{v_{i-1}}$ once in each direction. Thus (v_i, v_{i-1}) is also traversed by T_W, meaning that

W traverses the edge v_iv_{i-1}, from v_i to v_{i-1} after having reached v_i for the first time along $v_{i-1}v_i$. However, $v_i \neq v_0$ implying that W departs from v_i as often as it reaches v_i. On the other hand, W traverses some $e \in E_{v_i}$ no more than once by the choice of v_i; so there must be an $e' \in E_{v_i}$ not traversed by W from v_i. This contradicts **Steps 1** and **2** of the algorithm. It follows that our supposition by which W is not a bidirectional double tracing, is false. Thus the algorithm produces what it claims: a bidirectional double tracing. □

The similarities in the justifications of the above two algorithms is reflected in the following corollary which we state without proof.

Corollary 31. *If W is the final walk in G constructed by Trémaux's algorithm, then for any $v \in V(G) - \{v_0\}$, the last edge passsed by W from v is the first edge by which W reaches v. That is, W can be considered to have been constructed by Tarry's algorithm.*

We now combine the above two algorithms to arrive at another algorithm.

> **Hypothesis.** The hypotheses of Trémaux's and Tarry's algorithm are used simultaneously. Consequently, for
>
> $$E_{W,i} := \{e \in E_v \mid \lambda_W(e) = i\}$$
>
> defined upon reaching v in the course of constructing W, the sets $E_{W,0}, E_{W,1}$, and $E^{\text{in}}_{v,W} := E_{W,1} - E^{o}_{v,W}$ are known, where $E^{\text{in}}_{v,W}$ is the set of edges passed towards v but not from v.

Algorithm 10.

Define **Step 0** – **Step 4** as in Tarry's algorithm with the only exception that we choose e in **Step 1** with minimal $\lambda_W(e)$.

The working of Algorithm 10 is guaranteed by the working of Tarry's algorithm since the restriction concerning the choice of e in **Step 1** has no negative impact; that is, the final W is a bidirectional double tracing in G anyway. The relevance of Algorithm 10 lies in its application to eulerian graphs which is demonstrated by the next result. We state it without proof.

Theorem 32. *Let G be a connected eulerian graph, and let W be a bidirectional double tracing in G arising from an application of Algorithm 10. Let*

$$T = e_{i_1}, e_{i_2}, \ldots, e_{i_q}, \qquad q = E(G),$$

be the subsequence of W obtained by listing the edges of G according to their second occurrence in W. It follows that T is an eulerian trail of G.

We note in passing that for the case of plane trees, no local information is needed to produce a bidirectional double tracing, but a simple rule called the "right-hand-on-the-wall" rule, suffices (Left-handers are permitted to walk with their left hand on the wall).
Altough each of the Algorithms 8, 9 and 10 operates in $O(q)$, $q = |E|$, time, basing itself on the existence of bidirectional double tracings in connected graphs,

the question arises whether one can solve (MSP) such that every edge is traversed once and at most twice. In fact, if in addition to the local information of Algorithm 10, one operates with a counter (e. g. in the form of pencil and paper), Algorithm 10 can be improved.

> **Hypothesis.** In addition to the hypothesis of Algorithm 10 we assume that a function (the counter) $c : \{W\} \to \mathbb{N} \cup \{0\}$ is given, where $\{W\}$ denotes the set of walks produced algorithmically, such that $c(v_0) = 1$, and for W and $W' = W, e_i, v_{i+1}$ we have $|c(W) - c(W')| \leq 1$. We also use the sets defined in the hypothesis of Algorithm 10.

Algorithm 11. *A.S. Fraenkel's Algorithm*

This Algorithm also gives preference to untraversed edges as in Algorithm 10. The algorithm combines Algorithm 10 (except that Step 4 is replaced by **Outcome** below) and the counter c according to the following rules.

Rule 1 As long as $c(W) > 0$ proceed as in Algorithm 10.

Rule 2 The changing of c is defined in accordance with a certain property regarding v_{i+1} in $W' = W, e_i, v_{i+1}$

a) If $v_{i+1} \neq v_j$, $0 \leq j \leq i$, set $c(W') = c(W) + 1$.

b) If $v_{i+1} = v_j$ for some $j < i + 1$ and if $\left|E_{v_{i+1}} - E(W)\right| \geq 1$, while $|E_{W',0}| \leq 1$ for the same v_{i+1} set $c(W') = c(W) - 1$.

c) If neither a) nor b) applies, set $c(W') = c(W)$.

Rule 3 Suppose $c(W) = 0$. If $E_{W,0} \neq \emptyset$ at v_i, proceed as in Algorithm 10. Otherwise, set $e_i = e_{\text{in}}(v_i)$ if $v_i \neq v_0$, while the algorithm terminates for $v_i = v_0$.

Outcome The final W is a closed covering walk such that $\lambda_W(e) \leq 2$ for every $e \in E(G)$. If $\lambda_W(e) = 2$ for some $e \in E(G)$ then W uses e in both directions.

Justification. First of all, the counter c is a function as described in the hypothesis. For, c is increased by one if $v \in V(G)$ is visited by (the final) W for the first time. It is decreased by one at the same v if for the last time $E_{W,0} \neq \emptyset$ for this v. Expressed differently, after this departure from v, all edges in E_v are traversed at least once and c is not changed at v anymore. Whence at every vertex, c is increased by one and decreased by one precisely once. This increase and decrease of c is instantaneous if $d(v) \leq 2$. It follows that $c(W)$ is always a non-negative integer.
Suppose now that $c(W) = 0$ for some W starting at v_0 and ending at $v \in V(G)$; possibly $v_0 = v$. If $E_{W,0} \neq \emptyset$ at v, it follows from the above that no $e \in E_{W,0}$ joins v to any $x \in V(W)$; otherwise, c is not decreased at x which in turn implies $c(W) \neq 0$ at v, a contradiction. Similarly, $G_0 := \langle E(W) \rangle$ is an induced subgraph satisfying $E_x \subset E(G_0)$ for every $x \in V(G_0) - \{v\}$. Thus (see **Rule 1** and **Rule 3**) Algorithm 11 deviates from Algorithm 10 only after having passed

all edges of G, at which point the algorithm stops if $\{e_{\text{in}}(v)\} = \emptyset$ (i. e., $v = v_0$), or else reaches v_0 via the unique path $P(v, v_0)$ satisfying $E(P(v, v_0)) \subseteq \{e_{\text{in}}(v) \mid v \in V(G)\}$. It follows that Algorithm 11 constructs a closed covering walk such that the doubly traversed edges are used in both directions. □

The Chinese Postman Problem

Here, we follow in large parts the formulation of the topic as treated in Fleischner's book.[3]

Given a connected graph G, find a closed covering walk W in G of minimal length. This is, basically, how the *Chinese Postman Problem* (CPP) was originally formulated. Such a walk W will be called a *postman's tour* (PT for short). The idea of dealing with this problem arose at the end of the 1950's when Guan Meigu studied the following question. "A mailman has to cover his assigned segment before returning to the post office. The problem is to find the shortest walking for the mailman."

Whence the name Chinese Postman Problem; and it is quite common among Guan Meigu's colleagues worldwide to call him "The Chinese Postman".

Clearly, since every connected graph G has a closed covering walk W using every edge exactly twice (see Propositions 28, 29 and Corollary 30), the CPP is not an existence problem but a characterization problem concerning those closed covering walks which are postman's tours, and an algorithmic problem inasmuch as one seeks to determine a PT by a method which is as efficient as possible.

We shall now consider graphs G together with a cost function $c : E(G) \to \mathbb{R}^+$; $c(e)$ can be viewed as the "length" of the edge $e = xy$, or the time it takes to travel along e from vertex x to vertex y, or the costs connected with driving a vehicle from x to y along e, and so on. For $H \subseteq G$, an edge sequence $W = e_1, e_2, \ldots, e_r$ in G respectively, define

$$c(H) := \sum_{e \in E(H)} c(e), \quad c(W) := \sum_{i=1}^{r} c(e_i)$$

respectively.

> Thus the (general) CPP asks for a closed covering walk W in G such that $c(W)$ is minimal (CPP).

In this case we shall also speak of a postman's tour PT, PT(G) respectively. The following result has already been proved in principle, by Guan Meigu (for the case where $c(e) = 1$, $e \in E(G)$).

Theorem 33. *A closed covering walk W in the (connected) graph G with cost function $c : E(G) \to \mathbb{R}^+$ is a postman's tour if and only if W satisfies the following conditions:*

1) $1 \leq \lambda_W(e) \leq 2$ *for every* $e \in E(G)$*;*

[3]Herbert Fleischner, Eulerian Graphs and Related Topics, Annals of Discrete Mathematics, Vol. 50, ISBN: 978-0-444-89110-5

2) $c(H \cap C) \leq \frac{1}{2}c(C)$ *for every cycle* $C \subseteq G$, *where* $H = \langle\{e \in E(G) \mid \lambda_W(e) = 2\}\rangle$

Proof. Necessity. Suppose W is a PT. Transform G into a supergraph G^+ of G by adding, for every $e = xy \in E(G)$, precisely $\lambda_W(e) - 1$ edges joining x and y. Consequently, W is transformed into an eulerian trail T^+ of G^+; i. e., G^+ is eulerian. Extend the cost function c onto $E(G^+)$ by assigning the same value $c(e)$ to each of the additional $\lambda_W(e) - 1$ edges, $e \in E(G)$. Thus,

$$c(G^+) = c(T^+) = c(W).$$

Now, if $\lambda_W(e) > 2$ for any $e = xy \in E(G)$, then delete in G^+ two of these additional $\lambda_W(e) - 1$ edges to obtain a connected eulerian graph H^+. We have $G \subseteq H^+ \subset G^+$. For every eulerian trail T of H^+ we have

$$c(H^+) = c(T) < c(T^+) = c(W). \tag{1}$$

Since T corresponds to a closed covering walk W' in G satisfying $c(W') = c(T)$, and since W is a PT, we obtained a contradiction.
Now suppose for $H = \langle\{e \in E(G) \mid \lambda_W(e) = 2\}\rangle$ and for some cycle $C \subseteq G$ that

$$c(H \cap C) > c(C - (H \cap C)). \tag{2}$$

Construct G^+ as above and define H_1 by

$$E(H_1) := (E(H) - E(H \cap C)) \cup (E(C) - E(H \cap C)). \tag{3}$$

(note that $e \in E(C) - E(H \cap C)$ implies $\lambda_W(e) = 1$). Construct $G_1^+ \supseteq G$ from G by introducing an additional edge for every element of $E(H_1)$. G_1^+ is connected and, by (3), eulerian. Extend the cost function c onto $E(G_1^+)$ in the above manner. G_1^+ has an eulerian trail T_1 which corresponds to a closed covering walk W_1 of G, and

$$c(G_1^+) = c(T_1) = c(W_1) = c(W) - c(H \cap C) + c(C - (H \cap C)).$$

Whence we conclude from (2) that

$$c(W_1) < c(W),$$

a contradiction to W being a PT. Consequently, we must have

$$2c(H \cap C) \leq c(C - (H \cap C)) + c(H \cap C) = c(C),$$

i. e.,

$$c(H \cap C) \leq \frac{1}{2}c(C)$$

for every cycle $C \subseteq G$. That is, W satisfies condition 2) as well.
Sufficiency. Let W be any fixed PT and let W' be an arbitrary closed covering walk of G satisfying the conditions stated in the theorem. We have to show that $c(W) = c(W')$. Consider H and H' defined correspondingly. It follows from the definition of these graphs, and since W and W' are closed covering walks that

$$d_H(v) \equiv d_{H'}(v) \equiv d_G(v) (\text{mod } 2), \quad v \in V(G). \tag{4}$$

Thus, $H_0 := (H \cup H') - (H \cap H')$ is eulerian. H_0 need not be connected; however, by Theorem 25, H_0 has a cycle decomposition S_0 (possibly $S_0 = \emptyset$). Note that

$$c(W) = c(G) + c(H) = c(G) + c(H \cap H') + \sum_{C \in S_0} c(H \cap C),$$

$$c(W') = c(G) + c(H') = c(G) + c(H \cap H') + \sum_{C \in S_0} c(H' \cap C).$$

We conclude from the fact that W is a PT

$$c(W) - c(W') = \sum_{C \in S_0} (c(H \cap C) - c(H' \cap C)) \leq 0. \tag{5}$$

Suppose now $c(W) \neq c(W')$; thus (5) is a strict inequality; i. e.,

$$c(H \cap C_0) < c(H' \cap C_0) \quad \text{for some } C_0 \in S_0. \tag{6}$$

However, by definition of H_0

$$c(C_0) = c(H \cap C_0) + c(H' \cap C_0). \tag{7}$$

Thus we obtain from (6) and (7)

$$\frac{1}{2} c(C_0) < c(H' \cap C_0);$$

i. e., W' violates condition 2) of the theorem for $C_0 \in S_0$. This contradiction forces $c(W) = c(W')$. That is, W' is a PT. This finishes the proof of the theorem. □

Although Theorem 33 characterizes postman tours in a simple way, it is impractical from an algorithmic point of view since it requires, in principle, the examination of all cycles of G in the context of condition 2). Guan Meigu himself, being aware of this problem, showed that it suffices to examine chordless cycles when checking condition 2) for a closed covering walk W of G (of course, first one has to check whether W satisfies condition 1) of Theorem 33). But even this restriction to chordless cycle does not seem to be feasible for constructing a good algorithm to solve the CPP.
However, this theorem contains the starting point for what is needed to produce such a good algorithm. To this end, let us consider, for a postman's tour W in G, the graph

$$H = \langle \{e \in E(G) \mid \lambda_W(e) = 2\} \rangle$$

as defined in condition 2) of that theorem, together with the graph G^+ obtained from G by duplicating the elements of $E(H)$. Since G^+ is eulerian we have

$$d_H(v) \equiv d_G(v) (\text{mod } 2), \quad v \in V(G) \tag{4'}$$

(see (4) in the proof of Theorem 33, and define $d_H(v) = 0$ for $v \notin V(H)$). Moreover, H is acyclic because of condition 2) of the theorem., and the odd vertices of H are the odd vertices of G by the above congruence; let these vertices be denoted by $v_1, \ldots, v_{2k}$. Denote by $V_1(P)$ the endpoints of a path P.

It is left as an exercise to show that there are k edge-disjoint paths $P_1, \ldots, P_k$ in H such that

$$\bigcup_{i=1}^{k} V_1(P_i) = \{v_1, \ldots, v_{2k}\}.$$

On the other hand, it does not require knowledge of H to find such k paths $P_1, \ldots, P_k$; and if one has such k paths, then G_1^+ which is obtained from G by duplicating precisely the edges of these paths is a connected eulerian graph satisfying condition 1) (but possibly not condition 2)) of Theorem 33. However, if one knows H, it follows of necessity that H can be written as $H = \bigcup_{i=1}^{k} P_i$ since G^+ is eulerian and W is a PT.
Generalizing this idea of finding $P_1, \ldots, P_k$ such that

$$V_{\text{odd}} = V_{\text{odd}}(G) := \{v \in V(G) \mid d(v) = 1(\text{mod } 2)\} = \bigcup_{i=1}^{k} V_1(P_i)$$

we consider $\Pi_2(V_{\text{odd}})$, an arbitrary partition of V_{odd} into k classes of size 2, and denote by $\mathcal{P}_2(V_{\text{odd}})$ the set of all these partitions. For fixed $\Pi_2 \in \mathcal{P}_2(V_{\text{odd}})$ and arbitrary $\{v_{i_1}, v_{i_2}\} \in \Pi_2$, let P_{i_1,i_2} be a 'shortest' path (in terms of the cost function c) joining v_{i_1} and v_{i_2} in G, and denote

$$c_{i_1,i_2} = c(P_{i_1,i_2}).$$

Denote

$$c(\Pi_2) = \sum_{\{v_{i_1}, v_{i_2}\} \in \Pi_2} c_{i_1,i_2}.$$

With this notation we are led to an alternate characterization of the solutions of the CPP.

Theorem 34. *The problem of finding a solution of the CPP for the connected graph G with cost function $c : E(G) \to \mathbb{R}^+$ is equivalent to finding $\Pi_2^0 \in \mathcal{P}_2(V_{odd})$ such that*

$$c(\Pi_2^0) = min\{c(\Pi_2) \mid \Pi_2 \in \mathcal{P}_2(V_{odd})\}.$$

That is, if W is a PT in G,

$$c(W) = c(G) + c(\Pi_2^0).$$

The proof of Theorem 34 can be deduced easily from the preceding discussion and is therefore left as an exercise. We note, however, that the k paths corresponding to the elements of Π_2^0 are edge-disjoint; this follows from the minimality of $c(\Pi_2^0)$ and because $c(e) > 0$, $e \in E(G)$.
At a first glance, Theorem 34 does not seem to be an essentially better characterization of postman tours than Theorem 33. For, a simple combinatorial argument shows that for $k \geq 4$

$$|\mathcal{P}_2(V_{\text{odd}})| = (2k-1)(2k-3)\ldots 3.1 \geq 3^k$$

(The proof is left as an exercise). That is, if G has no even vertices, and $p := p_G \geq 8$,

$$|\mathcal{P}_2(V(G))| \geq 3^{\frac{p}{2}} > (1.7^p);$$

i. e., the number of partitions to be considered in determining $c(\Pi_2^0)$ grows, in principle, exponentially.
However, Theorem 34 points in the right direction. Using the notation introduced in the discussion preceding Theorem 34, we may obtain the following criterion.

Theorem 35. *Let G be a connected non-eulerian graph with cost function $c : E(G) \to \mathbb{R}^+$, and let c^* be the cost function of K_{2k} defined by*

$$c^*(v_i, v_j) = c_{i,j}, \quad v_i, v_j \in V_{odd}(G) = V(K_{2k}).$$

For $E_0 \subseteq E(K_{2k})$ define $c^*(E_0) := \sum_{e \in E_0} c^*(e)$.

A closed covering walk W of G is a PT if and only if

$$c(W) = c(G) + min\{c^*(L) \mid L \text{ is a 1-factor of } K_{2k}\}.$$

The proof of Theorem 35 follows from the observation that there is a 1-1-correspondence between the partitions $\Pi_2 \in \mathcal{P}_2(V_{\text{odd}})$ and the 1-factors L of K_{2k} and vice versa, and that $c(\Pi_2) = c^*(L)$ by the very definitions of $c(\Pi_2)$ and c^*.
Thus, the above discussion shows that the Chinese Postman Problem can be transformed into the following equivalent algorithmic form (CPP') which is more explicit than the original statement (CPP).

1) **Input:** Given the connected graph G with cost function $c : E(G) \to \mathbb{R}^+$, let $V_{\text{odd}}(G) = \{v_1, \ldots, v_{2k}\}$ denote the set of odd vertices of G, $k \geq 1$.

2) **SPP** (*Shortest Path Problem*): Determine $c_{i,j} = \text{min}c(P_{i,j})$, $1 \leq i < j \leq 2k$, where $P_{i,j}$ is an arbitrary path joining $v_i, v_j \in V_{\text{odd}}(G)$ in G. Store precisely one $P^*_{i,j}$ satisfying $c_{i,j} = c(P^*_{i,j})$, $1 \leq i < j \leq 2k$.

3) **MMP** (*Minimum Weight Perfect Matching Problem*): For K_{2k}, $V(K_{2k}) = V_{\text{odd}}(G)$, and for $c^* : E(K_{2k}) \to \mathbb{R}^+$ satisfying $c^*(v_iv_j) = c_{i,j}$, determine a 1-factor $L \subseteq E(K_{2k})$ such that $c^*(L)$ is minimal.

4) **EUL** (*Euler Tour Problem*): For every $v_iv_j \in L$ duplicate in G the edges of $P^*_{i,j}$ to obtain the connected eulerian graph G^+. Construct an arbitrary eulerian trail T^+ of G^+ and describe W, the closed covering walk of G corresponding to T^+. W is a PT.

This idea of transforming (CPP) into (CPP') is already contained basically in the work of J. Edmonds. As for the complexity of an algorithm based on (CPP'), we take note of the following facts:

SPP. The *Shortest Path Problem* stated in **2)** can be solved in polynomial time. For, if we apply Dijkstra's algorithm, say, for a fixed $v_i \in V_{\text{odd}}(G)$, we obtain in $O(p^2)$ time at the most not only all values $c_{i,j}$, but also corresponding paths $P^*_{i,j}$, $1 \leq j \leq 2k$, $j \neq i$. Thus, **SPP** can be solved in $2kO(p^2) \leq O(p^3)$ time at the most. Thus establishing K_{2k} with cost function c^* and a list of the above paths $P^*_{i,j}$ can be done in polynomial time $S(p)$.

MMP. The *Minimum Weight Perfect Matching Problem* (i. e., finding a 1-factor L in K_{2k} with minimum cost $c^*(L)$) can be solved in at most polynomial time $M(p)$.

EUL. G^+ can be constructed from G in polynomial time $E_1(P)$ since the paths $P^*_{i,j}$ corresponding to the elements of L can be found in polynomial time in the above list of all $P^*_{i,j}$'s. An eulerian trail T^+ in G^+ can be found in polynomial time $E_2(p)$, where T^+ can be transformed into W, a PT of G, in polynomial time $E_3(p)$. Thus, the Euler Tour Problem can be solved in polynomial time $E(p) := E_1(p) + E_2(p) + E_3(p)$.

Summarizing these considerations, we can say that the Chinese Postman Problem can be solved in polynomial time $C(p)$, where

$$C(p) = S(p) + M(p) + E(p).$$

Associating a variable x_e with every $e \in E(G)$, the original (CPP) can be expressed in the following equivalent form:

$$\left.\begin{array}{ll} \text{Determine} & x_e \in \{0,1\}, e \in E(G) \\ \text{such that} & \displaystyle\sum_{e \in E(G)} c(e)x_e \text{ is minimized} \\ \text{subject to} & \displaystyle\sum_{e \in E_v} (1 + x_e) \equiv 0 (\text{mod } 2), v \in V(G). \end{array}\right\} \text{(CPP")}$$

The equivalence of (CPP") and (CPP) follows from the fact that any postman's tour W of G corresponds to a solution of (CPP") satisfying

$$H = \langle \{e \in E(G) \mid x_e = 1\} \rangle$$

$$\text{and} \quad c(H) = \sum_{e \in E(G)} c(e)x_e = \sum_{e \in E(H)} c(e)x_e,$$

(where H is the subgraph of G defined by W in condition 2) of Theorem 33), and vice versa (also note that $c(W) = c(G) + c(H)$). Moreover, the congruences in (CPP") are equivalent to saying that G^+ obtained from G by duplicating the edges of H is eulerian.

List of Symbols

$\Delta(G)$	The maximum degree of the graph G.
$\delta(G)$	The minimum degree of the graph G.
$\kappa(G)$	The connectivity of the graph G.
$\kappa(x, y)$	The local connectivity of the vertices x and y.
$\lambda(G)$	The edge connectivity of a graph G.
$\lambda(x, y)$	The local edge connectivity of the vertices x and y.
$\lambda(xy)$	Number of edges joining the adjacent vertices x and y.
$\lambda_W(e)$	The number of uses of the edge e by a given walk W.
$\langle E_0 \rangle$	The subgraph induced by the edges of E_0.
$\langle V_0 \rangle$	The subgraph induced by the vertices of V_0.
Φ	The flow in a network.
$\rho(x, y)$	The maximum number of internally disjoint paths joining the vertices x and y.
$\rho_e(x, y)$	The maximum number of edge-disjoint paths joining the vertices x and y.
φ_a	The flow in arc a.
$A(D)$	The set of arcs of the digraph D.
A_v	The set of arcs incident to vertex v or incident from v.
A_v^+	The set of arcs incident from vertex v.
A_v^-	The set of arcs incident to vertex v.
$bc(G)$	The block-cutpoint graph of the graph G.
c_a	The capacity of arc a.
$d(v)$	The degree of vertex v.
$E(G)$	The set of edges of the graph G.
E_v	The set of edges incident to vertex v.
$id(v) = d^-(v)$	The number of arcs whose head is vertex v (in-degree).
$L(G)$	The line graph of the graph G.
$l(W)$	The length of the walk W.
$od(v) = d^+(v)$	The number of arcs whose tail is vertex v (out-degree).
$P(x, y)$	A path from vertex x to vertex y.

$S(G)$	The subdivision graph of G.
$V(G)$	The set of vertices of the graph G.
$W(x, y)$	A walk from vertex x to vertex y.

Index