Advanced and algorithmic graph theory

Master Study Mathematics

MAT.464UF

Summer term 2022

edited by

Joshua Erde and Philipp Sprüssel
Institute of Discrete Mathematics
TU Graz

Contents

Basic notations and facts about graphs

We write $\mathbb{N}$ for the set of non-negative integers and note, in particular, that we work under the convention that $0 \in \mathbb{N}$.

Definition (Graphs)**.** A *graph* is a pair $G = (V, E)$, where $V = V(G)$ is a set (called the *vertex set* of G) and $E = E(G)$ is a set of 2-element subsets of V (called the *edge set* of G). We write

$$|G| := |V(G)| \qquad \text{and} \qquad \|G\| := |E(G)|\,.$$

The elements of $V(G)$ and $E(G)$ are called *vertices* and *edges*, respectively. For an edge $\{u, v\}$, we will also use the short notation uv.

All graphs in this course are assumed to be finite, i.e., $|G|$, and thus also $\|G\|$, are finite.

Definition (Graph isomorphisms)**.** Two graphs G, H are *isomorphic* if there exists a bijection $f\colon V(G) \to V(H)$ such that

$$\forall u, v \in V(G), \qquad uv \in E(G) \iff f(u)f(v) \in E(H).$$

In words, vertices are adjacent in G if and only if their images under f are adjacent in H.

Unless otherwise stated, we consider graphs up to isomorphisms.

Definition (Adjacencies, neighbourhood, degrees)**.** Let G be a graph.

(i) Two vertices u, v of G are *adjacent* (or *neighbours*) if uv is an edge (of G). Two edges are *adjacent* if they share a vertex. Finally, a vertex and an edge are *incident* if the edge contains the vertex.

(ii) The set of all neighbours of a vertex v is called the *neighbourhood of* v and is denoted by $N(v)$. For a set $U \subset V(G)$, we define the *neighbourhood of* U to be the set

$$N(U) := \{v \in V(G) \setminus U\colon v \text{ has a neighbour in } U\} = \left(\bigcup_{u \in U} N(u)\right) \setminus U.$$

(iii) The *degree* of a vertex v, denoted by $d_G(v)$ (usually abbreviated as $d(v)$ when the graph G is clear from the context), is the number of edges incident with v, or equivalently, the size of the neighbourhood of v. Vertices of degree zero are called *isolated*.

(iv) The *minimum degree* $\delta(G)$, the *average degree* $d(G)$, and the *maximum degree* $\Delta(G)$ of G are defined by

$$\delta(G) := \min_{v \in V(G)} d(v), \quad d(G) := \frac{1}{|G|} \sum_{v \in V(G)} d(v), \quad \Delta(G) := \max_{v \in V(G)} d(v).$$

(v) If all vertices of G have degree r, then we call G *r-regular*. A 3-regular graph is also called *cubic*.

Proposition 0.1. *For every graph G, we have*

$$\delta(G) \leq d(G) \leq \Delta(G) \qquad \text{and} \qquad d(G) = \frac{2\,\|G\|}{|G|}.$$

In particular, the number of vertices in G with odd degrees is even.

Proof. The inequalities are trivial, and the second and third statement follows from the fact that each edge is counted exactly twice in $\sum_{v\in V(G)} d(v)$, i.e.,

$$\sum_{v\in V(G)} d(v) = \sum_{v\in V(G)} \sum_{\substack{e\in E(G):\\ e \text{ incident to } v}} 1 = \sum_{e\in E(G)} \sum_{\substack{v\in V(G):\\ v \text{ incident to } e}} 1 = \sum_{e\in E(G)} 2 = 2||G||,$$

which is sometimes known as the *handshaking lemma*. The second statement follows immediately, and the third since $\sum_{v\in V(G)} d(v) = 2||G||$ is even. □

Definition (Other notions of graphs)**.** Let V, E be sets.

(i) We call (V, E) a *multigraph* if E is a multiset whose elements are subsets of V of sizes 1 and 2. Edges that appear multiple times in E are *multiedges* (also *double edges*, *triple edges* etc. depending on the multiplicity), edges of size 1 are *loops*.

In multigraphs, degrees are usually defined as the number of incident edges, where loops *are counted twice*. Thus, in multigraphs, we have $d(v) \geq |N(v)|$, but in general not equality.

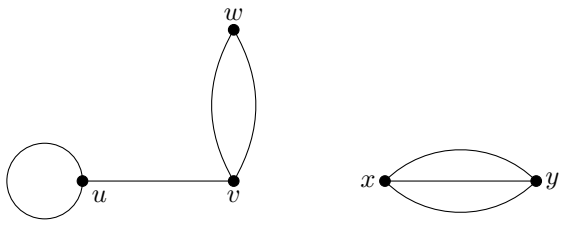

Figure 1: A representation of a multigraph on vertex set $\{u, v, w, x, y\}$. We have a loop at u, a double edge between v and w, and a triple edge between x and y.

(ii) If $E \subset V^2 \setminus \{(v, v) \colon v \in V\}$, then we call (V, E) a *directed graph*. We say that an edge (u, v) is directed *from u towards v*.

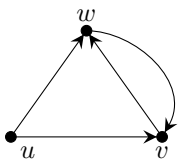

Figure 2: A representation of a directed graph on vertex set $\{u, v, w\}$.

In a directed graph D, one usually distinguishes between the *indegree* $d^-(v)$ of v, which is the number of edges towards v, and the *outdegree*

$d^+(v)$ of v, which is the number of edges away from v. Starting from these, one defines $\delta^-(D)$, $\delta^+(D)$, $d^-(D)$, $d^+(D)$, $\Delta^-(D)$, and $\Delta^+(D)$ as before.[1]

(iii) If $f\colon E \to \mathbb{R}$ is a function, then we call (V, E, f) a *weighted graph*. For every edge e, the value $f(e)$ is its *weight*.

Definition (Subgraphs)**.** Let G, H be graphs.

(i) H is called *subgraph* of G, which we write as $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. It is a *proper subgraph*, which we write $H \subsetneq G$, if at least one of the two relations is not an equality.

(ii) A subgraph H of G is *spanning* if $V(H) = V(G)$.

(iii) A subgraph H of G is *induced* if $E(H) = E(G) \cap \binom{V(H)}{2}$.

(iv) Given subgraphs H_1, H_2 of G, we write

$$H_1 \cup H_2 := (V(H_1) \cup V(H_2), E(H_1) \cup E(H_2))$$
$$\text{and} \qquad H_1 \cap H_2 := (V(H_1) \cap V(H_2), E(H_1) \cap E(H_2)).$$

Definition (Paths, cycles, complete graphs)**.** Let $n \in \mathbb{N}$.

(i) A *path of length* n is a graph P on vertices $v_0, \dots, v_n$, for which $v_iv_j \in E(P)$ if and only if $i - j = \pm 1$. We write $P = v_0 \dots v_n$.

For a path P and a vertex v of P, we write Pv for the subpath of P until (and including) v. Analogously, vP denotes the subpath of P starting at v.

(ii) If A and B are vertex sets in a graph G, an *A–B path* in G is a path $P = v_0 \dots v_n$ that is a subgraph of G and satisfies $V(P) \cap A = \{v_0\}$ and $V(P) \cap B = \{v_n\}$. In particular, if $v_0 \in A \cap B$, then the path $P = v_0$ of length zero is an A–B path.

If H is a subgraph of G, then an *H-path* (in G) is a path in G of length at least one that shares only its end vertices with H.

(iii) For $n \geq 3$, a *cycle of length* n is a graph C on vertices $v_1, \dots, v_n$, for which $v_iv_j \in E(C)$ if and only if $i - j = \pm 1 \bmod n$.

(iv) A *complete graph* is a graph with $E = \binom{V}{2}$. By K^n we denote the complete graph on n vertices.

In order to keep notations simple, we shall omit set brackets whenever possible; for instance we will write a–b path instead of $\{a\}$–$\{b\}$ path.

Definition (Complete and independent sets)**.** Let G be a graph, $U \subseteq V(G)$, and $F \subseteq \binom{V(G)}{2}$.

(i) By $G[U]$ we denote the induced subgraph $\left(U, E(G) \cap \binom{U}{2}\right)$ of G. We say that $G[U]$ is the graph that G *induces on* U.

[1] Easy exercise: Show that $d^-(D) = d^+(D)$.

(ii) U is called *complete* (or a *clique*) if $G[U]$ is complete, i.e., if the vertices of U are pairwise adjacent.

(iii) U is *independent* (or *stable*) if $G[U]$ is edgeless, i.e., if no two vertices of U are adjacent.

(iv) F is *independent* if its elements are pairwise disjoint, i.e., if no two edges in F are adjacent.

We also write $G - U$ for the graph $G[V(G) \setminus U]$ obtained by deleting all vertices in the set U (and all edges incident with at least one such vertex) and write $G - F$ and $G + F$ for the graphs $(V(G), E(G) \setminus F)$ and $(V(G), E(G) \cup F)$, respectively.

Definition (Bipartite graphs)**.** A graph G is called *bipartite* if its vertex set can be partitioned into two independent sets. This partition is not necessarily unique, but when a bipartite graph G is given, we will usually assume that it comes with a partition into parts A and B.

Given $s, t \in \mathbb{N}_{>0}$, we define the *complete bipartite graph*

$$K_{s,t} := (A \cup B, A \times B),$$

where A, B are disjoint and $|A| = s$, $|B| = t$.

Proposition 0.2. *A graph is bipartite if and only if it does not contain a cycle of odd length.*

Proof. Exercise. □

Definition (Contraction, minor, subdivision)**.** Let G be a graph.

(i) For a vertex set S in G, we say that the graph G/S defined by

$$\begin{aligned} V(G/S) &:= V(G - S) \cup \{S\}, \\ E(G/S) &:= E(G - S) \cup \{vS \colon \exists s \in S \text{ with } vs \in E(G)\} \end{aligned}$$

is obtained from G by *contracting* S. We also say that G/S is a *contraction* of G. For disjoint vertex sets $S_1, \dots, S_k$, we define the contraction as

$$G/(S_1, \dots, S_k) := \Big(\big((G/S_1)/\dots\big)/S_k\Big).$$

Observe that the result does not depend on the order of the sets.

(ii) If $S_1, \dots, S_k$ are disjoint *connected* vertex sets (called *branch sets*) in G, then every subgraph of $G/(S_1, \dots, S_k)$ is called a *minor* of G. In particular, every subgraph of G is also a minor of G.

(iii) We say that G is a *subdivision* of H if we can obtain G from H by replacing each edge $e \in E(H)$ by a path of length at least one. In that case, the vertices of G that correspond to vertices of H are called *branch vertices*.

If G has a subgraph that is a subdivision of H, we call H a *topological minor* of G.

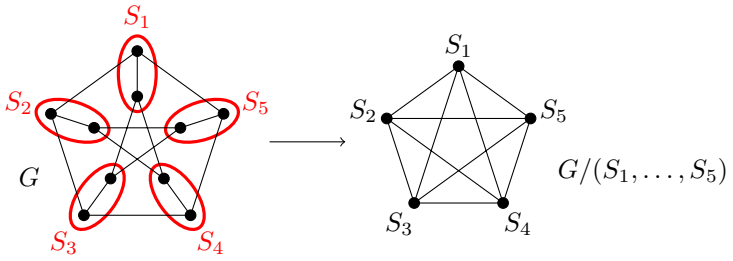

Figure 3: Disjoint connected vertex sets $S_1, \ldots, S_5$ in a graph G and the resulting contraction $G/(S_1, \ldots, S_5) = K^5$. Every subgraph of K^5 is thus a minor of G.

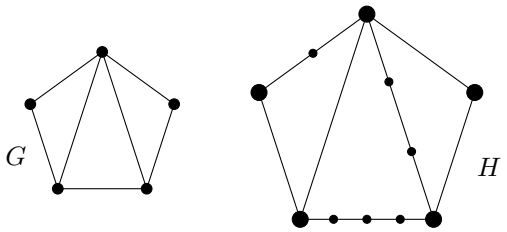

Figure 4: A graph G and a subdivision H of G. The branch vertices are drawn thick. G is a topological minor of any graph that contains H as a subgraph.

Algorithms and running time

We will compare the running times, i.e., the numbers of operations, of the algorithms in this course with respect to the size of the input graph G, i.e., $|G|$ and $\|G\|$. To simplify notation, we shall use the variables n and m instead of $|G|$ and $\|G\|$, respectively, whenever we talk about running time. We will usually be interested in the *order* of the running time, that is, we disregard constant factors.

Definition (Landau notation)**.** If $f, g\colon \mathbb{N} \to \mathbb{R}_{>0}$ are functions, we write

- $f(n) = O(g(n))$ if there exists a constant $c^+ > 0$ such that $f(n) \leq c^+ g(n)$ for all (sufficiently large) n;
- $f(n) = \Omega(g(n))$ if there exists a constant $c^- > 0$ such that $f(n) \geq c^- g(n)$ for all (sufficiently large) n;
- $f(n) = \Theta(g(n))$ if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

For functions with multiple variables, the notation is analogous.

In some cases, we will be able to express the running time in the form $\Theta(g(n))$, which means that the running time has the same order for all graphs, independent of their structure. In other cases, some graphs will have a much longer running time than others, even if both have the same numbers of vertices and edges. In that sense, the notation $O(g(n))$ is a *worst case* description, which means that an algorithm that, for instance, needs exponential time for a small fraction of all graphs and linear time for all others, will be considered "worse" or "slower" than an algorithm that needs, say, quadratic time for *all* graphs.

Remark. *A smaller order running time does not necessarily mean that an algorithm is faster in practice. An algorithm with running time*

$$10^{10^{10^{10}}} n = \Theta(n)$$

has smaller order running time than an algorithm with running time

$$10^{-10^{10^{10}}} n^2 = \Theta(n^2),$$

but the latter algorithm will be much faster for all "sensible" input sizes.

Chapter 1

Connectivity

Definition (Connected graphs, components)**.** Let G be a non-empty graph.

(i) G is called *connected* if it contains a u–v path for all $u, v \in V(G)$.

(ii) A set $U \subseteq V(G)$ is *connected* (in G) if $G[U]$ is connected.

(iii) If U is connected and maximal (with respect to inclusion) with this property, then we call $G[U]$ a *component* of G.

Definition (Forests, trees)**.** A graph without cycles is called *forest*. A connected forest is a *tree*. Thus, the components of a forest are trees. A vertex of degree one in a forest is called *leaf*.

Theorem 1.1. *The following statements are equivalent for a non-empty graph T.*

(i) T is a tree;

(ii) for all vertices $u, v \in V(T)$, T contains a unique u–v path;

(iii) T is minimally connected, i,e., T is connected, but for every edge $e \in E(T)$, $T - e$ is not connected;

(iv) T is maximally cycle-free, i,e., T contains no cycles, but for every $e \in \binom{V(T)}{2} \setminus E(T)$, $T + e$ contains a cycle;

(v) T is connected and $\|T\| = |T| - 1$;

(vi) if $|T| \geq 2$, then T has at least two vertices of degree one, and for every such vertex v, $T - v$ is a tree.

Proof. Exercise. □

1.1 Spanning trees

Definition (Spanning trees)**.** A *spanning tree* of a graph G is a tree that is a spanning subgraph of G.

Proposition 1.2. *Every connected graph G has a spanning tree.*

Proof. G is connected, and so contains a connected spanning subgraph. Let T be a minimally connected spanning subgraph of G. By Theorem 1.1, T is a tree and thus a spanning tree of G. □

Proposition 1.2 gives a pure existence statement. The natural question arising from this is how to find a spanning tree of a given connected graph. Or more generally: If G is any graph (not necessarily connected), describe an algorithm that finds the components of G and produces a spanning tree of each component.

There is a obvious 'local' way to extend a tree T in G to a larger one: Given a vertex u in T, we search for a vertex in $N(u)$ which is not in T and add the edge uv (and vertex v) to T. Clearly if we repeat such a step, we will eventually build a spanning tree of G, if one exists, however in order to give an algorithm we will need to choose an *order* in which to make these local extensions. There are two fundamentally different classical ways to solve this problem, called *breadth first search* and *depth first search*, which illustrate the difference between two common data structures, a *queue* and a *stack*.

In breadth first search (BFS), we keep track of the vertices whose neighbours we will explore in a queue Q, a *first in, first out* data structure. We start at some given vertex u, which we call the *root*, and initially $Q = \{u\}$. We first find all of the neighbours of u and add them one by one to the back of the queue, adding the edges from u to these vertices to our tree T. Once there are no more neighbours left to find, we delete u from the queue.

We now move on the the next vertex v in the queue. We find all the neighbours of v which are not already in T and add them to the back of the queue, adding the edges from u to these vertices to T. Once there are no more neighbours left to find, we delete v from the queue. We repeat this step until the queue is empty. Note that the vertices which are added to the queue first, have their neighbourhoods searched first.

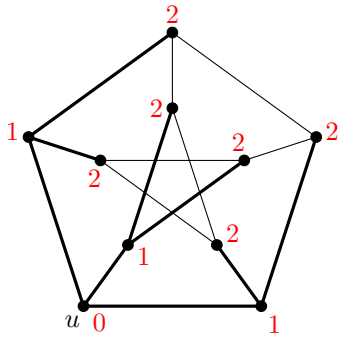

Figure 1.1: Finding a spanning tree (bold) in a connected graph via breadth first search. The red numbers are the layers of the vertices.

A simplified program implementing this strategy looks as follows.

```
PROCEDURE BFS(G,u)
BEGIN
    known[u] := TRUE;
    layer[u] := 0;
    Queue Q := (u);
    For (v ∈ V(G) − u){
        known[v] := FALSE;
        layer[v] := ∞;
    }
    WHILE (Q ≠ ∅){
        v := First(Q);
        FOR (w ∈ N(v)){
            IF (known[w] = FALSE){
                known[w] := TRUE;
                Concatenate (Q,w);
                layer[w] := layer[v] + 1;
                parent[w] := v;
            }
        }
        Remove v from Q;
    }
END
```

If we want to explore all components of G, we need to incorporate the BFS procedure in a main program that recursively finds a yet undiscovered vertex v and starts exploring the component containing v by calling BFS(G,v). The running time of such a program can be seen to be $\Theta(n+m)$.

Proposition 1.3. *Suppose that we run* BFS(G,u) *for a graph G and $u \in V(G)$.*

(i) The vertices $w \in V(G)$ with known$[w]$ = TRUE *are precisely the ones that lie in the component C of G that contains u.*

(ii) The graph $T = (V(C), E_{BFS})$ with

$$E_{BFS} := \{\{w, \text{parent}[w]\} \colon w \in V(C-u)\}$$

is a spanning tree of C.

(iii) The value layer$[w]$ *equals the distance of u and w, that is, the length of the shortest u–w path in G (where vertices with no path between them—i.e. vertices in different components—have infinite distance by convention).*

Proof. It will be useful to first prove the following three claims:

Firstly, suppose that at some point in the algorithm $Q = (v_0, v_1, \ldots, v_r)$, then we claim that

$$\text{layer}[v_0] \leq \text{layer}[v_1] \leq \ldots \leq \text{layer}[v_r] \leq \text{layer}[v_0] + 1.$$

This condition is trivially true at the start of the algorithm. Now each time the inner loop runs the element v_0 is removed from the queue and some neighbours $w_1, \ldots, w_i$ (potentially none) of v_0 are added to the back of the queue, where $\text{layer}[w_i] = \text{layer}[v_0] + 1$, and so the condition remains true.

Secondly, we claim that if v is removed from Q before w is, then $\text{layer}[v] \leq \text{layer}[w]$. Clearly it is sufficient to prove the statement when v and w are removed

consecutively. In this case, either w is already in the queue when we run the inner loop for $N(v)$, in which case layer$[v] \leq$ layer$[w]$ by the above claim, or v is the only element in Q when we run the inner loop for $N(v)$, and w is added to Q in this step, and so layer$[w] =$ layer$[v] + 1$.

Finally, we claim that if known$[w]$=TRUE, then layer$[w] \geq d(u, w)$. Indeed, since $(w, \text{parent}[w]) \in E(G)$ for all such w, and the layer of parent$[w]$ is one smaller than the layer of w, it follows that $w, \text{parent}[w], \text{parent}[\text{parent}[w]] \ldots$ is a path of length layer$[w]$ which ends at the unique vertex u with layer 0.

We can now prove (iii), that layer$[w] = d(u, w)$ for all w with known$[w]$ =TRUE. Note that, $d(u, w) < \infty$ if and only if $w \in V(C)$ and layer$[w]$ is defined if and only if known$[w]$ =TRUE and so, in particular, (i) follows from (iii).

Let w be a vertex with $d(u, w)$ minimal such that layer$[w] \neq d(u, w)$, where by the above we may assume that layer$[w] > d(u, w)$. Let v be a neighbour of w with $d(u, v) = d(u, w) - 1$, and note that by assumption

$$\text{layer}[w] > d(u, w) = d(u, v) + 1 = \text{layer}[v] + 1. \tag{1.1}$$

Since known$[v]$ =TRUE, at some point v is added to Q and since the algorithm terminates when Q is empty, at some point $v = \text{First}(Q)$ and we run the inner loop for $N(v)$.

If known$[w]$ =FALSE at this point, then during the loop we set layer$[w] =$ layer$[v] + 1$, contradicting (1.1).

If known$[w]$=TRUE, and w has already been deleted from Q, then w was deleted from Q before v and so layer$[w] \leq$ layer$[v]$, again contradicting (1.1).

Finally, if known$[w]$=TRUE and w is still in Q, then both v and w are in Q at the same time, and so layer$[w] \leq$ layer$[v] + 1$, again contradicting (1.1).

For (ii) we first note that, since $X = C(V)$, parent$[w]$ is defined for all $w \in V(C-u)$ and so T is well-defined. Furthermore, since every vertex $w \in V(C)-u$ is connected to its parent, and the layer of the parent is strictly smaller than the layer of the vertex, it follows that T contains a path from each vertex to the unique vertex u with layer$[u] = 0$. In particuarly, T is connected.

Furthermore, if we orient each edge of T from w to parent$[w]$, then each vertex has outdegree one in this orientation. In particular, if T contains a cycle, then it must be oriented cyclically. However, since each oriented edge goes to a vertex with strictly smaller layer, such a cycle cannot exist. It follows that T is a tree, and T spans $V(C)$ by construction. □

In contract, in depth first search (DFS) we keep track of the vertices whose neighbourhoods we will explore in a stack S, a *first in, last out* data structure.

More concretely, we start with some given root vertex u and with a stack $S = \{u\}$. We find some neighbour v of u and add v to the front of the stack, and add the edge uv to our tree T.

We now consider the first vertex v in the stack. If v has no neighbours outside of T, we delete v from the stack. Otherwise, we find some neighbour w of v outside of T and add w to the front of the stack, and add the edge vw to our tree T. We repeat this step until the stack is empty. Note that the vertices which are added to the queue last, have their neighbourhoods searched first.

Again, we can write this approach down in a simple program. However, rather than explicitly building a stack to keep track of the order we search the neighbourhoods, we will instead give a *recursive* algorithm, and the order in

which we search the neighbourhoods will be determined by an *implicit stack*, given by the order that the subroutines of the algorithm are called.

```
PROCEDURE DFS(G,u)
BEGIN
   t := t + 1;
   DFSnum[u] := t;
   FOR (v ∈ N(u)){
      IF (known[v] = FALSE){
         known[v] := TRUE;
         parent[v] := u;
         DFS(G,v);
      }
   }
END
```

This procedure needs a main program that sets the initial values and starts the procedure in each component.

```
BEGIN
   FOR (v ∈ V(G)){
      DFSnum[v] := 0;
      known[v] := FALSE;
      parent[v] := v;
   }
   t := 0;
   FOR (v ∈ V(G)){
      IF (known[v] = FALSE){
         known[v] := TRUE;
         DFS(G,v);
      }
   }
END
```

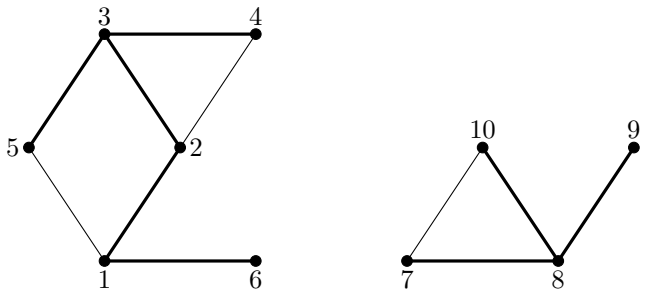

Figure 1.2: Exploring a graph via depth first search. The numbers are the values of DFSnum, the bold edges are the edges from vertices to their parent.

We note that, as with BFS, the main program for DFS constructs a spanning tree for each component of a (not necessarily connected) graph G in time $\Theta(n+m)$.

Proposition 1.4. *Suppose that G is connected. Let $u \in V(G)$ and suppose that we run* DFS(G,u) *(with the initial values provided by the main program).*

(i) The graph $T = (V(G), E_{DFS})$ *with*

$$E_{DFS} := \{\{v, \text{parent}[v]\} \colon v \in V(G) - u\}$$

is a spanning tree of G.

(ii) DFS(G,u) *has running time* $\Theta(n + m)$.

(iii) Suppose that $vw \in E(G)$ *and* $\text{DFSnum}[v] < \text{DFSnum}[w]$. *Let* $P = v_0 \dots v_k$ *be the unique* v–w *path in* T, *then* $v_{i-1} = \text{parent}[v_i]$ *for all* $i \in \{1, \dots, k\}$.

Proof. (i) T is a spanning subgraph of G by definition (as long as it is well-defined). We first note that, since G is connected, DFS(G, v) is run precisely once for each vertex $v \in V(G)$ during the run of the algorithm. Indeed, it runs at most once, since known[v] is set to be TRUE when DFS(G, v) runs, and conversely it is easy to show by induction on $d(u, v)$ that DFS(G, v) runs for each $v \in V(G)$. In particular, parent[v] is defined for all $v \in V(G) - u$, and so T is well-defined, and DFSnum[v] is set for all $v \in V(G)$.

For $t = 1, \dots, |G|$, denote by V_t the set of vertices with $\text{DFSnum}[v] \leq t$. First observe that $|V_t| = t$ (the algorithm does not stop until the whole graph is explored) and thus in particular $V_{|G|} = V(G)$. We prove by induction that $T[V_t]$ is a tree for all t. The induction basis is clear, because V_1 is a single vertex. For the induction step, observe that the unique vertex in $V_{t+1} \setminus V_t$ has degree one in $T[V_{t+1}]$ and thus the implication (v) $\Rightarrow$ (i) of Theorem 1.1 proves that $T[V_{t+1}]$ is a tree, because $T[V_t]$ is.

(ii) We run DFS(G,v) exactly once for each vertex v; during this run, we check each neighbour of v. Thus, every vertex of G appears at least once and every edge of G is checked exactly twice, once from each of its end vertices. This implies that the running time is $\Theta(n + m)$.

(iii) Suppose that $\text{DFSnum}[v] = t$. If $t = 1$ (i.e. $v = u$), then it is clear that P has the desired property. Indeed, since $\text{DFSnum}[\text{parent}[x]] < \text{DFSnum}[x]$ for all $x \neq u$, for each vertex $x \neq u$ the path $u, \text{parent}[u], \text{parent}[\text{parent}[u], \dots$, which is contained in T, must terminate at u and, since T is a tree, this is the unique $u - x$ path in T by Theorem 1.1 (ii).

If $t > 1$, consider the time when DFS(G,v) runs. All vertices in V_{t-1} will be rejected by the IF-loop. Thus, if C is the component of $G - V_{t-1}$ that contains v (and hence also w), then DFS(G,v) has the same output as DFS(C,v) would have, only the values of DFSnum are shifted by $t - 1$. Since v is the root of the tree produced by DFS(C,v), P has the desired property. □

Remark. *Both BFS and DFS find spanning trees in linear time. The advantage of BFS is that it also finds the shortest paths from the root to any other vertex. The main benefit of DFS is the additional information from Proposition 1.4(iii) about the edges of the graph which are not in the spanning tree. We will see a possible way to exploit this knowledge later in the course.*

In many applications, the graph G (which we now assume to be connected) will be weighted, where the weights on the edges correspond to some 'cost

function'. In that situation, the goal is to find a 'cheapest' spanning tree, that is, a spanning tree of smallest total weight. For this problem, we have several different algorithms.

Alg. 1. Start with $T_1 = (V(G), \emptyset)$ and recursively add edges as follows. Among all edges $e \in E(G) \setminus E(T_1)$ for which $T_1 + e$ is cycle-free, choose one with the smallest weight and add it to T_1. The algorithm ends when no more edges can be added.

Alg. 2. Start with $T_2 = G$ and recursively delete edges as follows. Among all edges $e \in E(T_2)$, for which $T_2 - e$ is connected, choose one of largest weight and delete is from T_2. The algorithm ends when no more edges can be deleted.

While these two algorithms are 'global' in the sense that the information about all edges is used in every step, there are also ways to construct a spanning tree with only 'local' information.

Alg. 3. For some vertex v, start with $T_3 = (\{v\}, \emptyset)$ and recursively add vertices and edges as follows. Among all edges from $V(T_3)$ to $V(G) \setminus V(T_3)$, pick one with smallest weight and add it (together with its end vertex in $V(G) \setminus V(T_3)$) to T_3. The algorithm ends when T_3 spans G.

Our last algorithm only works if we assume all weights in the graph to be distinct.

Alg. 4. Start with $T_4 = (V(G), \emptyset)$ and recursively add edges as follows. For every component H of T_4, among all edges from $V(H)$ to $V(G) \setminus V(H)$, pick one with smallest weight. We then add all these edges to T_4 simultaneuosly. The algorithm ends when T_4 is connected.

If, say, edges uv, vw, and wu have the same weight, then Algorithm 4 can create a cycle within the first step.

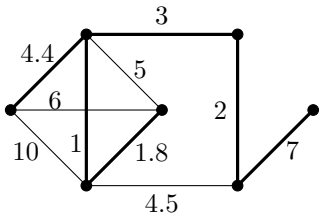

Figure 1.3: A weighted graph and its spanning tree of minimal total weight. Check that each of the four algorithms results in this spanning tree.

Proposition 1.5. *Algorithms 1–3 produce a spanning tree of minimal total weight. If all edge weights are distinct, then there is a unique such spanning tree, and Algorithm 4 produces this spanning tree as well.*

Proof. We prove the result for Algorithms 3 and 4; the proofs for the other two algorithms are similar and left as an exercise. We start with Algorithm 4, under the additional condition that all weights are distinct.

By construction, T_4 is spanning and connected when the algorithm stops, it remains to prove that it is also acyclic. In fact, it will be useful to prove a slightly stronger statement: We claim that for any cycle C of G if e is the (unique!) edge of C with maximal weight, then e is not in T_4. Note that, in particular, the claim implies that T_4 is acyclic.

To prove the claim, let C be a cycle in G and let e be the edge of maximal weight in C. Suppose that e is not in T_4 at some time during Algorithm 4. Let H be a component of T_4 (at that time) that contains an end vertex of e. If e is an edge from $V(H)$ to $V(G) \setminus V(H)$, then there is some edge $e' \in C - e$ that also goes from $V(H)$ to $V(G) \setminus V(H)$, see Figure 1.4. Since e' has smaller weight than e, the algorithm does not add e to T_4 in this step. Since e is not in T_4 at the beginning of the algorithm, it is never added to T_4. Therefore, T_4 is cycle-free and thus a spanning tree.

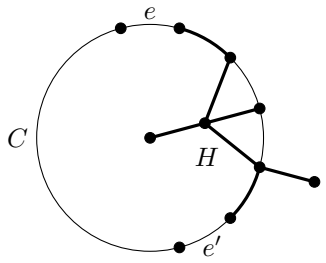

Figure 1.4: A cycle C in G and a component H of T_4 at an intermediate stage of the algorithm. If e has the largest weight on C and is not yet in T_4, then it will not be added in the next step (and thus will never be added).

We shall now prove that T_4 has strictly smaller total weight than all other spanning trees of G. This will prove both the uniqueness of the spanning tree with smallest total weight and the fact that Algorithm 4 constructs this spanning tree. Among all spanning trees of G, let T be one with smallest total weight. If $T \neq T_4$, then T_4 contains edges that are not in T, since all trees on the same vertex set have the same number of edges by the implication (i) $\Rightarrow$ (v) of Theorem 1.1. Let $e \in E(T_4) \setminus E(T)$. Then $T + e$ contains a cycle C, because T was maximally cycle-free, by implication (i) $\Rightarrow$ (iv) of Theorem 1.1.

Let e' be the edge of C with the largest weight and consider the graph $T' := T + e - e'$, see Figure 1.5. We removed an edge from a cycle of $T + e$, hence T' is connected. Now the implication (v) $\Rightarrow$ (i) of Theorem 1.1 shows that T' is a spanning tree. Furthermore, as we have seen above, e' is not in T_4 and thus in particular $e' \neq e$. Therefore, e' has larger weight than e. This implies that T' has smaller total weight than T, a contradiction to the choice of T.

Let us now consider Algorithm 3. The graph T_3 is a tree throughout the algorithm, because we start with a tree and add a vertex of degree one in each step, which results in a tree by the implication (vi) $\Rightarrow$ (i) of Theorem 1.1. Since the algorithm only ends when T_3 spans G, T_3 is a spanning tree.

It remains to prove that T_3 has smallest total weight, even if there are edges with identical weights. Let $e_1, e_2, \ldots, e_{n-1}$ be the edges in T_3, in the order which they were added to T_3. Suppose that T is a spanning tree of G of smallest total

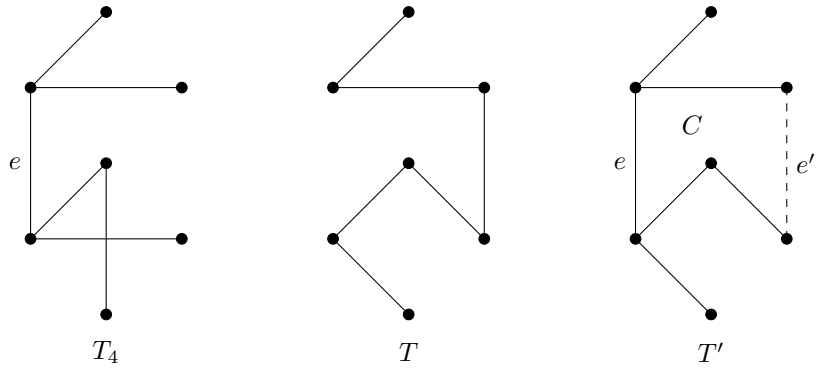

Figure 1.5: Constructing a tree of smaller total weight if $T \neq T_4$.

weight, which contains as large an initial segment of the sequence $e_1, e_2, \ldots, e_{n-1}$ as possible. We will show that $T = T_3$, and hence T_3 has smallest total weight.

Indeed, if $T \neq T_3$ then let i be minimal such that $uv := e_i \notin E(T)$. Consider the point at which we added e_i to T_3. At this stage in the algorithm there is some vertex set S such that e_i is an edge of minimal weight between S and $V(G) \setminus S$. Since T is a spanning tree and $e_i \notin E(T)$, $T + e_i$ contains a cycle C, and C must contain an even number of edges between S and $V(G) \setminus S$. Hence there is some edge $f \neq e_i$ such that $f \in E(C)$ and f is an edge between S and $V(G) \setminus S$. Hence, by our assumption on e_i it follows that e_i has weight at most as large as f.

However, then $T' = T + e_i - f$ is a spanning whose weight is at most that of T, and hence, by our assumption on T, T' is also a spanning tree of minimal weight. Furthermore, since we added the edges $e_1, \ldots, e_{i-1}$ to T_3 before e_i, it follows that the endvertices of $e_1, e_2, \ldots, e_{i-1}$ are all contained in S, and so f is not equal to one of $e_1, e_2, \ldots, e_{i-1}$. Hence, T' contains $e_1, e_2, \ldots, e_i$, contradicting our assumption that T was a smallest weight spanning tree containing the largest initial segment of $e_1, e_2, \ldots, e_{n-1}$.

□

1.2 k-connectedness

Definition (k-connectedness)**.** Let $k \in \mathbb{N}$. A graph G is called *k-connected* if $|G| > k$ and for every set $U \subset V(G)$ with $|U| < k$, the graph $G - U$ is connected.

In other words, a graph with more than k vertices is k-connected if we have to delete at least k vertices to disconnect it. Clearly, every non-empty graph is 0-connected, so this is not a particularly interesting property, and 1-connectedness is equivalent to connectedness, provided that the graph has at least two vertices.

Definition (Connectivity)**.** For every non-empty graph G, we define the *connectivity* of G to be

$$\kappa(G) := \max\{k \in \mathbb{N} \colon G \text{ is } k\text{-connected}\}.$$

Our definition of connectivity is the answer to the question "How many vertices do we have to delete in order to disconnect the graph?" On the other

hand, 1-connectedness is equivalent to asking for one path between each pair of vertices. It would thus be natural to assume that k-connectedness should be defined by the existence of k paths between each pair of vertices. We shall shortly see that this property is in fact equivalent to our definition.

Definition (Internally disjoint paths)**.** We say that paths $P_1, \dots, P_k$ are *internally disjoint* if, for all $i \neq j$, the intersection $P_i \cap P_j$ consists solely of common end vertices of P_i and P_j (in particular, the paths do not share edges).

Theorem 1.6. *Let $k \in \mathbb{N}$. A graph G with at least two vertices is k-connected if and only if for all distinct vertices $u, v \in V(G)$, there are k internally disjoint u–v paths in G.*

We shall prove Theorem 1.6 with the help of a much more general result. In order to formulate this result, we need two more notations.

Definition (A–B separator)**.** Given sets $A, B \subseteq V(G)$, we say that a set S of vertices or edges *separates* A and B if every A–B path in G meets S. If S is a set of vertices, we also call it an *A–B separator*.

Definition (a–B fan)**.** Suppose that $a \in V(G)$ and $B \subseteq V(G) \setminus \{a\}$. If $P_1, \dots, P_k$ are a–B paths that meet only in a, we say that these paths form an *a–B fan*.

Theorem 1.7 (Menger 1927)**.** *Let G be a graph.*

(i) For any $A, B \subseteq V(G)$, the maximum number of pairwise disjoint A–B paths equals the size of a smallest A–B separator.

(ii) For all distinct vertices $a, b \in V(G)$ with $ab \notin E(G)$, the maximum number of internally disjoint a–b paths equals the size of a smallest a–b separator in $V(G) \setminus \{a, b\}$.

(iii) For all $a \in V(G)$ and $B \subseteq V(G) \setminus \{a\}$, the largest number of paths in an a–B fan equals the size of a smallest a–B separator in $V(G) \setminus \{a\}$.

The three statements in Theorem 1.7 are also called the "set version", the "vertex version", and the "fan version" of Menger's theorem.

Proof of Theorem 1.6 from Theorem 1.7. The statement is trivial for $k = 0$ and $k = 1$, thus we may assume that $k \geq 2$.

"$\Longrightarrow$". Let G be k-connected and suppose, for contradiction, that there are vertices $u \neq v$ that are not linked by k internally disjoint paths. By Theorem 1.7(ii), this implies that $uv \in E(G)$. Then $G' := G - uv$ has at most $k - 2$ internally disjoint u–v paths and thus contains a u–v separator $S' \not\ni u, v$ with $|S'| \leq k - 2$ by Theorem 1.7(ii). As G is k-connected, we have $|V(G')| = |V(G)| > k$ and thus $G' - S' - u - v$ contains some vertex w. Now S' separates w from u or from v in G, say from u. But then $S' \cup \{v\}$ is a u–w separator in G of size at most $k - 1$, contradicting the fact that G is k-connected.

"$\Longleftarrow$". For any $S \subset V(G)$ with $|S| < k$ and any two vertices $u, v \in G - S$, the set S cannot meet all k internally disjoint u–v paths, hence $G - S$ is connected and thus G is k-connected if we know that $|G| > k$. To prove this, let $u \neq v$ be any two vertices and choose internally disjoint u–v paths $P_1, \dots, P_k$, ordered (increasingly) by their length. While P_1 might be just an edge, each other P_i has to have length at least two and thus contains some vertex $w_i \notin \{u, v\}$. Now $u, v, w_2, \dots, w_k$ are $k + 1$ distinct vertices; in particular, $|G| > k$. □

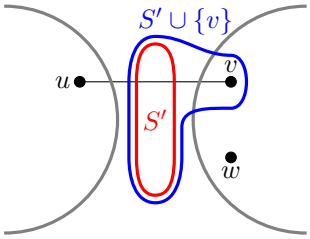

Figure 1.6: Finding a small separator if u, v are not linked by k internally disjoint paths.

Proof of Theorem 1.7. We first prove that all three statements are equivalent.

"(i) $\Rightarrow$ (ii)". Let a, b be given as in (ii). Fix a largest set $\mathcal{P}_{a,b}$ of internally disjoint a–b paths and a smallest a–b separator S in $V(G) \setminus \{a, b\}$.

Set $A := N(a)$ and $B := N(b)$. Any a–b path contains a unique A–B path and vice versa, any A–B path can be extended to an a–b path by adding one edge to a and b, respectively. Thus, $\mathcal{P}_{a,b}$ corresponds to a largest set $\mathcal{P}_{A,B}$ of disjoint A–B paths, and both sets contain the same number of paths.

Since each a-b separator in $V(G)$ which contains neither a nor b is also an A-B separator, S is an A–B separator. Furthermore, since each a–b path contains an A–B path, each A–B separator is also an a-b separator. It follows that S is also a smallest A–B separator. Hence, by (i), we have $|\mathcal{P}_{A,B}| = |S|$ and thus $|\mathcal{P}_{a,b}| = |S|$, as desired.

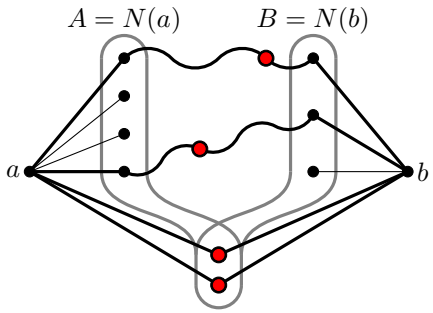

Figure 1.7: Deriving the vertex version of Menger's theorem from the set version. Any largest set of internally disjoint a–b paths (bold) corresponds to a largest set of disjoint A–B paths, while any smallest A–B separator (red vertices) is also a smallest a–b separator in $V(G) \setminus \{a, b\}$.

"(ii) $\Rightarrow$ (iii)". Let a, B be given as in (iii) and let $\mathcal{P}_{a,B}$ and S be a largest a–B fan and a smallest a–B separator in $V(G) \setminus \{a\}$, respectively.

Add a new vertex b to the graph and connect it to all vertices in B to obtain a new graph G'. Analogously to the first implication, $\mathcal{P}_{a,B}$ corresponds to a largest set $\mathcal{P}_{a,b}$ of internally disjoint a–b paths, while S is a smallest a–b separator in $V(G') \setminus \{a, b\}$. By construction, $ab \notin E(G')$ and thus we can apply (ii) to G' to deduce that $|\mathcal{P}_{a,b}| = |S|$ and thus $|\mathcal{P}_{a,B}| = |S|$, as desired.

"(iii) $\Rightarrow$ (i)". Let A, B be given as in (i). Add a vertex a to the graph and connect it to all of A. Like in the previous case, (i) follows.

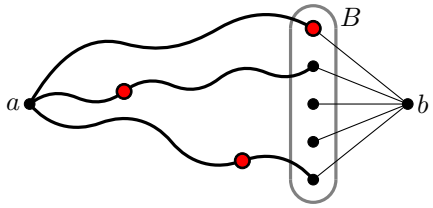

Figure 1.8: Deriving the fan version of Menger's theorem from the vertex version.

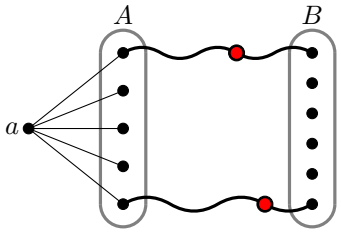

Figure 1.9: Deriving the set version of Menger's theorem from the fan version.

It remains to prove one of the three statements; we shall prove (i). Let S be a smallest A–B separator. We know that there are *at most* $k := |S|$ disjoint A–B paths, because S contains a vertex in every A–B path; we thus need to find k such paths. Let $\mathcal{P}$ be a set of disjoint A–B paths. We say that a set $\mathcal{Q}$ of $|\mathcal{P}|+1$ disjoint A–B paths *extends* $\mathcal{P}$ if

$$\left(A \cap V\left(\bigcup \mathcal{P}\right)\right) \subsetneq \left(A \cap V\left(\bigcup \mathcal{Q}\right)\right)$$
$$\text{and} \qquad \left(B \cap V\left(\bigcup \mathcal{P}\right)\right) \subsetneq \left(B \cap V\left(\bigcup \mathcal{Q}\right)\right),$$

where $\bigcup \mathcal{P}$ denotes the union of all paths in $\mathcal{P}$. We aim to prove that if $\mathcal{P}$ has less than k elements, then it can be extended; this statement will then clearly imply (i). We prove this claim for fixed G and A (but for all B simultaneuosly) by induction on the number $|\bigcup \mathcal{P}|$ of vertices in (the paths in) $\mathcal{P}$.

Set $l := |\mathcal{P}| < k$. The set $B \cap \bigcup \mathcal{P}$ is too small to separate A and B and thus $G - (B \cap \bigcup \mathcal{P})$ contains an A–B path R. If R avoids all paths in $\mathcal{P}$, then $\mathcal{Q} := \mathcal{P} \cup \{R\}$ extends $\mathcal{P}$. (In particular, this is the case when $\mathcal{P} = \emptyset$, thus proving the induction basis.) We may thus assume that R contains vertices in $\bigcup \mathcal{P}$. Let x be the last such vertex on R and let P be the path in $\mathcal{P}$ that x lies on. By the choice of R, we have $x \notin B$.

Set $B' := B \cup V(xP \cup xR)$ and $\mathcal{P}' := (\mathcal{P} \setminus \{P\}) \cup \{Px\}$. We know that $|\mathcal{P}'| = l < k$ and $|\bigcup \mathcal{P}'| < |\bigcup \mathcal{P}|$. Moreover, A and B' cannot be separated by less than k vertices, because each such separator would also separate A and B. Thus, we can apply the induction hypothesis to $\mathcal{P}'$ and obtain a set $\mathcal{Q}'$ of $l+1$ disjoint A–B' paths that extends $\mathcal{P}'$. This in particular means that all end vertices of paths in $\mathcal{P}'$ are also end vertices of paths in $\mathcal{Q}'$. Thus, one path $Q \in \mathcal{Q}'$ ends in x, one path Q' ends in a vertex $y \in B' \setminus V(\bigcup \mathcal{P}')$, and the other $l-1$ paths end in $\left(V(\bigcup \mathcal{P}') \cap B'\right) \setminus \{x\}$. If $y \in B \setminus (V(P) \cup V(R))$, then

$$\mathcal{Q} := (\mathcal{Q}' \setminus \{Q\}) \cup \{QxP\}$$

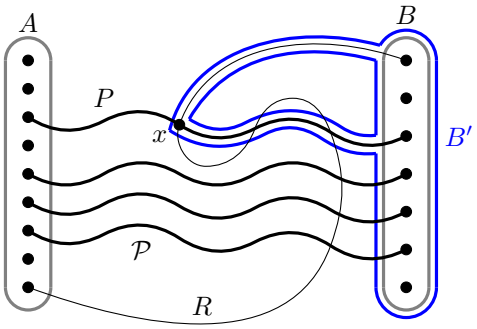

Figure 1.10: How the vertex x and the set B' arise from a path system $\mathcal{P}$ (bold) and a path R.

is an extension of $\mathcal{P}$. Otherwise, we have $y \in V(xP \cup xR) \setminus \{x\}$. If $y \in V(xP)$, then

$$\mathcal{Q} := (\mathcal{Q}' \setminus \{Q, Q'\}) \cup \{QxR, Q'yP\}$$

is the desired extension of $\mathcal{P}$, otherwise the extension is

$$\mathcal{Q} := (\mathcal{Q}' \setminus \{Q, Q'\}) \cup \{QxP, Q'yR\}.$$

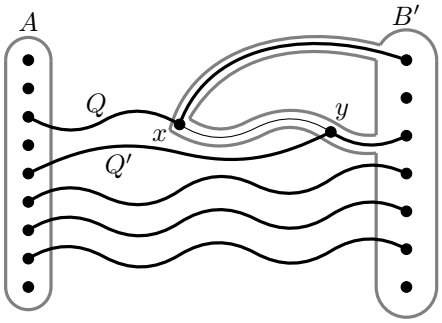

Figure 1.11: Extending the paths $Q, Q' \in \mathcal{Q}'$ in the case $y \in V(xP)$.

Either way, we have found an extension of $\mathcal{P}$, which finishes the proof of (i) and thus the proof of Theorem 1.7. □

1.3 2-connected graphs

Let us look at the special case $k = 2$.

Proposition 1.8. *Let G be a graph. Then the following statements are equivalent.*

(i) G is 2-connected;

(ii) $|G| \geq 2$ and any two distinct vertices lie on a common cycle;

(iii) G has no isolated vertices, $\|G\| \geq 2$, and any two distinct edges lie on a common cycle.

Proof. "(i) $\Rightarrow$ (iii)". If G is 2-connected, then it has at least three vertices, none of which is isolated. In particular, G has at least two edges. Let uv and xy be distinct edges and set $A := \{u, v\}$ and $B := \{x, y\}$. As G is 2-connected, the smallest A–B separator has size two and thus the set version of Menger's theorem implies that there are two disjoint A–B paths P and Q. Then $(P \cup Q) + uv + xy$ is the desired cycle.

"(iii) $\Rightarrow$ (ii)". $|G| \geq 2$ follows from the fact that G has an edge. Since every edge of G lies in a cycle, there are no vertices of degree one and thus every vertex has degree at least two. For distinct vertices u, v, we can thus pick distinct edges e_u and e_v incident with u and v, respectively. The cycle containing both e_u and e_v that is provided by (iii) also contains both u and v.

"(ii) $\Rightarrow$ (i)". Any two vertices that lie on a common cycle are also linked by two internally disjoint paths. Thus, (i) follows directly from (ii) and Theorem 1.6. □

2-connected graphs can also be characterised by the existence of a so-called "ear-decomposition".

Definition (Ear-decomposition)**.** An *ear-decomposition* of a graph G is a sequence $C, P_1, \ldots, P_k$ of subgraphs of G ($k = 0$ is allowed) such that

(i) C is a cycle;

(ii) each P_i is a $(C \cup P_1 \cup \cdots \cup P_{i-1})$-path;

(iii) $G = C \cup P_1 \cup \cdots \cup P_k$.

The paths $P_1, \ldots, P_k$ are called *ears*.

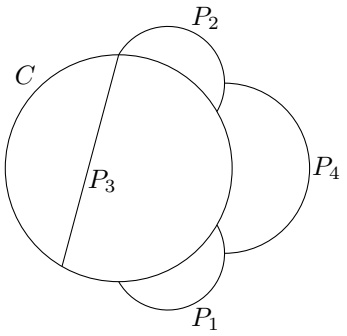

Figure 1.12: An ear-decomposition.

Proposition 1.9. *A graph G is 2-connected if and only if it has an ear-decomposition.*

For the proof of Proposition 1.9, we need an auxiliary result.

Lemma 1.10. *If H_1, H_2 are non-empty subgraphs of G, then*

$$\kappa(H_1 \cup H_2) \geq \min\{\kappa(H_1), \kappa(H_2), |H_1 \cap H_2|\}.$$

Proof. Denote the minimum of the three values on the right hand side by μ. Clearly, $H_1 \cup H_2$ has at least $|H_1| \geq \kappa(H_1) + 1 \geq \mu + 1$ vertices. If we delete less than μ vertices, then the remainders of both H_1 and H_2 will be connected and have non-empty intersection, thus $H_1 \cup H_2$ remains connected. □

Proof of Proposition 1.9. "$\Longleftarrow$". Let $C, P_1, \ldots, P_k$ be an ear-decomposition of G. We prove by induction over k that G is 2-connected. This is trivial for $k = 0$. Now suppose that $k \geq 1$ and that every graph with an ear-decomposition involving less than k paths is 2-connected. This means that in particular the subgraph $H := C \cup P_1 \cup \cdots \cup P_{k-1}$ of G is 2-connected. The path P_k has two vertices u, v in H; let C' be the union of P_k and some u–v path in H. Then $G = H \cup C'$ is 2-connected by Lemma 1.10, because both H and C' are 2-connected and their intersection contains at least two vertices (u and v).

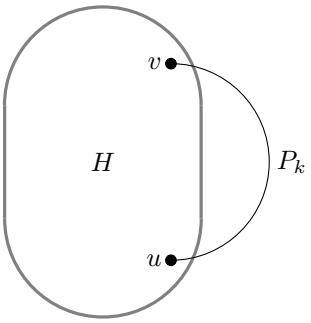

Figure 1.13: H and P_k intersect in two vertices, thus each component of $P_k - v$ meets $H - v$ in at least one vertex.

"$\Longrightarrow$". If G is 2-connected, then by Proposition 1.8, G contains a cycle and thus in particular a subgraph that has an ear-decomposition. Let H be a (containment-wise) largest such subgraph with ear-decomposition $C, P_1, \ldots, P_k$. If there were a vertex $v \in V(G) \setminus V(H)$, then apply the fan version of Menger's theorem (i.e. Theorem 1.7(iii)) to v and $V(H)$ to find a v–$V(H)$ fan consisting of two paths. The union P of these paths is an H-path and thus $H \cup P$ has an ear-decomposition (namely $C, P_1, \ldots, P_k, P$), a contradiction to the choice of H. Thus, H spans G.

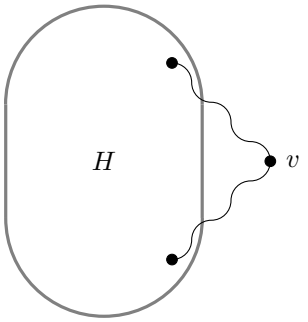

Figure 1.14: Finding a larger subgraph with an ear-decomposition if H is not spanning.

But now every edge in $E(G)\setminus E(H)$ would be an H-path, again contradicting the maximality of H. Therefore we have $H = G$, which means that G has an ear-decomposition. □

In a similar manner as the components of a graph decompose it into connected subgraphs, one can define a decomposition into 2-connected subgraphs.

Definition (Cut vertex, bridge, block)**.** A vertex v in a graph G is called *cut vertex* if $G-v$ has more components than G. In other words, v is a cut vertex if it separates two of its neighbours. An edge e of G is called *bridge* if it separates its end vertices. A *block* of G is a subgraph H of G that is maximal with the property that it is connected and has no cut vertex.[1]

By the definition of 2-connectedness, we can characterise 2-connected graphs via the non-existence of cut vertices.

Proposition 1.11. *A connected graph with at least three vertices is 2-connected if and only if it has no cut vertices.* □

What do the blocks of a graph G look like?

Remark. *If a block B has at least three vertices, then it is 2-connected by Proposition 1.11. By Proposition 1.8, such a block does not contain a bridge of G, because a bridge is not contained in any cycle (why?). Vice versa, each non-bridge lies in some cycle (why?) and thus in some 2-connected block. This means that the blocks of G are*

- *the maximal 2-connected subgraphs of G,*
- *all bridges (including their end vertices) in G, and*
- *all isolated vertices of G.*

We see that the blocks of G do *not* necessarily partition G. For instance, a tree has a block for each of its edges (because each edge is a bridge) and thus some vertices will lie in several blocks. But we shall see that the blocks partition the *edge set* of G.

Lemma 1.12. *Let G be a graph.*

(i) Any two distinct blocks of G intersect in at most one vertex.

(ii) Each vertex of G lies in at least one block and each edge lies in precisely one block.

(iii) Two edges lie in the same block if and only if they lie on a common cycle.

(iv) A vertex lies in more than one block if and only if it is a cut vertex.

Proof. (i) Suppose, for contradiction, that there are distinct blocks $B_1 \neq B_2$ which intersect in (at least) two vertices u, v. Since both B_1 and B_2 contain two vertices, they are not isolated vertices, and if B_1 were a single edge then $B_1 \subsetneq B_2$, contradicting either the maximality of the blocks as subgraphs. A similar argument holds for B_2, and hence we may assume that B_1 and B_2 are 2-connected. But then $B_1 \cup B_2$ is 2-connected by Lemma 1.10, contradicting the maximality of B_1 and B_2.

[1] The graph H has no cut vertex, but it may contain vertices that are cut vertices of G.

(ii) Each vertex/edge of G forms a connected subgraph without a cut vertex and thus is contained in some block. By (i), blocks are edge-disjoint.

(iii) If two edges e_1, e_2 lie on a common cycle C, then C is 2-connected and thus contained in a block B, which in turn contains e_1 and e_2.

Vice versa, if $e_1 \neq e_2$ lie in the same block B, then B has at least three vertices and thus is 2-connected. By Proposition 1.8, e_1 and e_2 lie on a common cycle in B, and so in particular in G.

(iv) Suppose that v lies in the intersection of two distinct blocks B_1, B_2. Since neither block is a subset of the other, neither of the two blocks is an isolated vertex. Let u_1 and u_2 be neighbours of v in B_1 and in B_2, respectively. If there exists a u_1–u_2 path in $G - v$, then v, u_1, u_2 lie on a common cycle C. Since C is a connected graph without a cut vertex it is contained in some block B_3. However, since $u_1v \in B_1 \cap B_3$ it follows from (ii) that $B_1 = B_3$, and similarly since $u_2v \in B_2 \cap B_3$ it follows that $B_2 = B_3$, contradicting the distinctness of B_1 and B_2. Thus, there is no u_1–u_2 path in $G - v$, meaning that v is a u_1–u_2 separator and thus in particular a cut vertex.

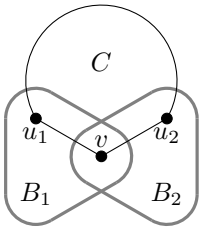

Figure 1.15: Finding a cycle C if v is not a u_1–u_2 separator.

Vice versa, suppose that v is a cut vertex and let u_1, u_2 be two of its neighbours that are separated by v. Let B_1 and B_2 be the blocks that contain the edges vu_1 and vu_2, respectively. We have $B_1 \neq B_2$, because otherwise this block would have to be 2-connected, which is not possible, as v separates u_1 and u_2. Thus, v lies in at least two blocks. □

Definition (Block-cut-vertex-graph)**.** Given a graph G, let $\mathcal{B}$ and $\mathcal{C}$ be the sets of its blocks and its cut vertices, respectively. The *block-cut-vertex-graph* is the graph $\mathrm{bc}(G)$ with vertex set $\mathcal{B} \cup \mathcal{C}$ and edge set

$$\{Bc\colon B \in \mathcal{B}, c \in \mathcal{C}, \text{ and } c \in V(B)\}.$$

Proposition 1.13. *Let G be a graph.*

(i) $\mathrm{bc}(G)$ *is a forest.*

(ii) $\mathrm{bc}(G)$ *is a tree if and only if G is connected.*

Proof. (i) Clearly, $\mathrm{bc}(G)$ is bipartite with sides $\mathcal{B}$ and $\mathcal{C}$, hence any cycle would alternately consist of blocks and cut vertices. Thus, a shortest cycle in $\mathrm{bc}(G)$ would have the form $C_{\mathrm{bc}} = c_1B_1c_2B_2 \dots c_kB_kc_1$. Since the vertices in C_{bc} are distinct, and distinct blocks share at most one

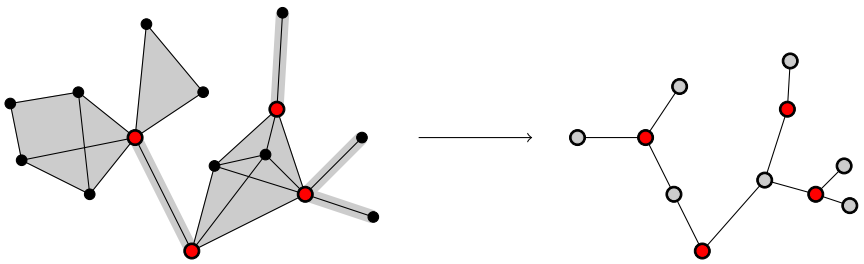

Figure 1.16: A connected graph and its block-cut-vertex-graph. Blocks are marked in gray, while cut vertices are drawn red.

vertex, it follows that $B_i \cap B_j = c_i$ if $j = i - 1 \bmod k$. Suppose for a contradiction that $B_i \cap B_j$ contains a vertex for some $1 \leq i < j \leq k$ with $i - j \not\equiv \pm 1 \bmod k$. In this case it is clear that $vB_ic_{i+1}B_{i+1} \dots c_jB_jv$ would be a cycle in $\mathrm{bc}(G)$ shorter than C_{bc}, a contradiction.

For each $1 \leq i \leq k$, let P_i be a c_i–c_{i+1} path in B_i (where we write $c_{k+1} := c_1$). Then, by the above, $P_1, \dots, P_k$ are internally disjoint and thus $C := P_1 \cup \dots \cup P_k$ is a cycle in G through $c_1, \dots, c_k$. But then $C \cup B_1 \cup \dots \cup B_k$ would be 2-connected by Lemma 1.10, a contradiction to the maximality of the B_i's. Therefore, $\mathrm{bc}(G)$ is cycle-free, i.e. a forest.

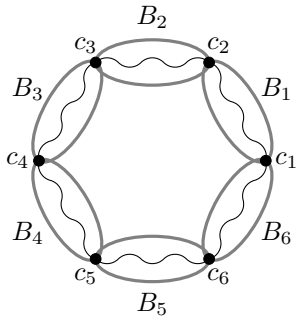

Figure 1.17: How a cycle in $\mathrm{bc}(G)$ gives rise to a cycle in G.

(ii) By (i), we only need to prove that $\mathrm{bc}(G)$ is connected if and only if G is connected.

Suppose that $\mathrm{bc}(G)$ is connected and let $u, v \in V(G)$. By Lemma 1.12 (ii) there are blocks $B_u \ni u$ and $B_v \ni v$. Since $\mathrm{bc}(G)$ is connected there is some path $B_1v_1B_2v_2 \dots v_{k-1}B_k$ in $\mathrm{bc}(G)$ with $B_1 = B_u$ and $B_k = B_v$, where this path may be a single vertex if $B_u = B_v$. For each $2 \leq i \leq k$, since B_i is connected, there is a $v_{i-1} - v_i$ path P_i in B_i and there is an $u - v_1$ path P_u in B_u and a $v_k - v$ path P_v in B_u. Then $\bigcup_{i=2}^{k} P_i \cup P_u \cup P_v$ is a connected subgraph of G which contains u and v, and hence G contains a $u - v$ path. It follows that G is connected.

Conversely, suppose that G is connected. If G contains no cut-vertices, then it has a unique block, and so $\mathrm{bc}(G)$ is trivially connected, or both

or both $\mathcal{B}$ and $\mathcal{C}$ are non-empty. In which case, for any block B and cut-vertex c, since G is connected, there is a path P in G from c to B. If we let $c = c_1, c_2, \ldots, c_k$ be the cut-vertices in P in order, and B_i be the blocks in which the paths $c_i P c_{i+1}$ are contained in (Exercise: Show any path with doesn't contain any cut-vertices in contained in a block). Note that, in this case, $c_k P \subseteq B$. It is clear then that $cB_1c_2B_2 \ldots B_{k-1}c_kB$ is an $c-B$ path in $\mathrm{bc}(G)$. Since each vertex in $\mathcal{C}$ is connected to each vertex in $\mathcal{B}$, and both are non-empty, it follows that $\mathrm{bc}(G)$ is connected. □

An obvious question is whether it is possible to determine the blocks of a graph algorithmically, preferably with low running time. To answer this question, let us take a look at depth first search again.

Definition (Edges queried by DFS)**.** Run the DFS algorithm for a graph G. This produces a spanning tree for every component of G; let T be their union.

(i) An edge of T is called *tree edge*.

(ii) Let $e = uv$ be an edge of G. At some point in the algorithm, the FOR-loop of DFS(G,u) is run with argument v. We say that e *is queried from* u at that time. Analogously, we define the time when e is queried from v.

(iii) An edge is called *forwards edge* if it queried first from its end vertex with smaller value of DFSnum[·], otherwise we call it *backwards edge*.

Lemma 1.14. *The non-tree edges are precisely the backwards edges.*

Proof. Let $e = uv$ be a non-tree edge and suppose that e is queried first from v. When e is queried from v, we have known[u] = TRUE, because otherwise e would have become a tree edge at that time, and hence DFS(G, u) has already started. However, since e has not been queried from u, DFS(G, u) has not finished, and hence DFS(G, v) was called as a subroutine whilst DFS(G, u) was running. It follows that DFSnum[u] < DFSnum[v].

Conversely, if $e = uv = \mathrm{parent}[v]v$ is a tree edge then e is queried at u at the point when parent[v] = u is set, and immediately after that DFS(G, v) is started. Hence e is queried first from u, and clearly

$$\mathrm{DFSnum}[u] < \mathrm{DFSnum}[\mathrm{parent}[v]] = \mathrm{DFSnum}[v].$$

□

Lemma 1.14 shows that the DFS algorithm partitions $E(G)$ into the set E_T of tree edges and the set E_B of backwards edges.

Definition (Ancestors, descendants)**.** Direct each edge of G as $e = (u, v)$, where u is the vertex from which e is queried first.

For vertices v, w of G, we say that v is an *ancestor* of w if $v \neq w$ and the unique v–w path in T is directed from v to w. We call v a *descendant* of w if $v = w$ or w is an ancestor of v.

By T_v we denote the graph that T induces on the set of descendants of v. By definition, all vertices of T_v are connected to v by a path in T_v, hence T_v is connected and thus, being a subgraph of T, a tree.

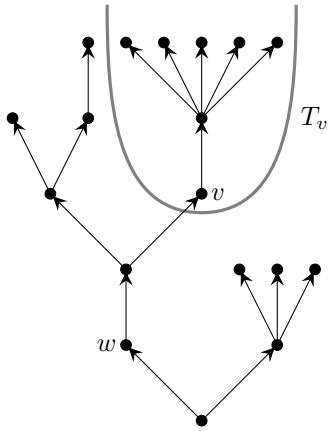

Figure 1.18: A DFS tree with directed edges. The vertex v is a descendant of w and, vice versa, w is an ancestor of v. Each vertex is also a descendant of itself.

Corollary 1.15. *For every non-tree edge (u, v), v is an ancestor of u.*

Proof. Since (u, v) is a non-tree edge and $e = uv$ was queried first from u, by Lemma 1.14 DFSnum$[u] >$ DFSnum$[v]$. Hence, by Proposition 1.4(iii) the unique $u - v$ path in T is given by $u, \text{parent}[u], \text{parent}[\text{parent}[u]], \ldots$. In particular, since all tree edges are oriented as $(\text{parent}[x], x)$, it follows that u is a descendent of v, and so v is an ancestor of u. □

When is a vertex a cut vertex? Suppose that $u = \text{parent}[v]$. If there are no backwards edges from T_v to $G - T_v - u$, then u separates $V(T_v)$ and $V(G - T_v - u)$ and thus u is a cut vertex (unless $G - T_v - u$ is empty).

Definition (Low point)**.** For each vertex $v \in V(G)$, we define the *low point* of v as

$$\text{LowPoint}[v] := \min\big\{\{\text{DFSnum}[v]\} \cup \{\text{DFSnum}[y] \colon \exists x \in V(T_v) \colon (x, y) \in E_B\}\big\}.$$

In other words, the low point of v is the smallest DFSnum one can reach from v by following the edges of G according to their directions, while using *at most one backwards edge.*

If we want to determine all low points in G efficiently, we can make use of the observation that each backwards edge (x, y) with $x \in V(T_v)$ is either of the type (v, y) or satisfies $x \in V(T_w)$ with w a child of v (that is, $v = \text{parent}[w]$).

Remark. *For every $v \in V(G)$, we have*

$$\begin{aligned}\text{LowPoint}[v] = \min\big\{&\{\text{DFSnum}[v]\} \cup \{\text{DFSnum}[y] \colon (v, y) \in E_B\} \\ &\cup \{\text{LowPoint}[w] \colon (v, w) \in E_T\}\big\}.\end{aligned}$$

Let us now modify the DFS algorithm so that it also determines tree edges, backwards edges, and low points.

```
PROCEDURE DFSb(G,u)
BEGIN
   t := t + 1;
   DFSnum[u] := t;
   LowPoint[u] := DFSnum[u];
   For (v ∈ N(u)){
      IF (known[v] = FALSE){
         known[v] := TRUE;
         parent[v] := u;
         E_T := E_T ∪ {(u, v)};
         DFSb(G,v);
         LowPoint[u] := min{LowPoint[u], LowPoint[v]};
      }
      ELSE IF (v ≠ parent[u]){
         E_B := E_B ∪ {(u, v)};
         LowPoint[u] := min{LowPoint[u], DFSnum[v]};
      }
   }
END
```

The main program has to be changed accordingly so as to set the initial values $E_B = E_T = \emptyset$ (and to run DFSb instead of DFS). Observe that the running time still has the same order as before.

Definition (Leading edge)**.** An edge $(v, w) \in E_T$ is called *leading edge* if LowPoint$[w] \geq$ DFSnum$[v]$. This is the case if and only if every backwards edge that starts in T_w ends either in T_w or in v.

Lemma 1.16. *Let H be a component of G and denote the root of $T[V(H)]$ by s. Let $v \in V(H) \setminus \{s\}$, $u :=$ parent$[v]$, and $(v, w) \in E(H)$. Then the following statements are equivalent.*

(i) uv and vw lie in the same block of G.

(ii) (v, w) is not a leading edge.

Proof. "(i) $\Rightarrow$ (ii)". If uv and vw lie in the same block, then they lie on a common cycle C by Lemma 1.12(iii). Now either (v, w) is a backwards edge (and thus in particular not a leading edge) or C contains an edge from T_w to $G - T_w - v$, which means that (v, w) is not a leading edge.

"(ii) $\Rightarrow$ (i)". Suppose that (v, w) is not a leading edge. If it is a backwards edge, then w is an ancestor of u and thus the unique w–u path in T, together with the edges uv and vw, forms a cycle containing both uv and vw. If (v, w) is a tree edge, then the fact that (v, w) is not a leading edge implies that there is a descendant x of w and an ancestor z of v such that (x, z) is a backwards edge. Now the union of xz and the unique z–x path in T is a cycle that contains both uv and vw.

Either way, uv and vw lie on a common cycle and thus in the same block by Lemma 1.12(iii). □

The following lemma shows that leading edges can be used to identify cut vertices.

Lemma 1.17. *For a component H of G, let s be the root of $T[V(H)]$.*

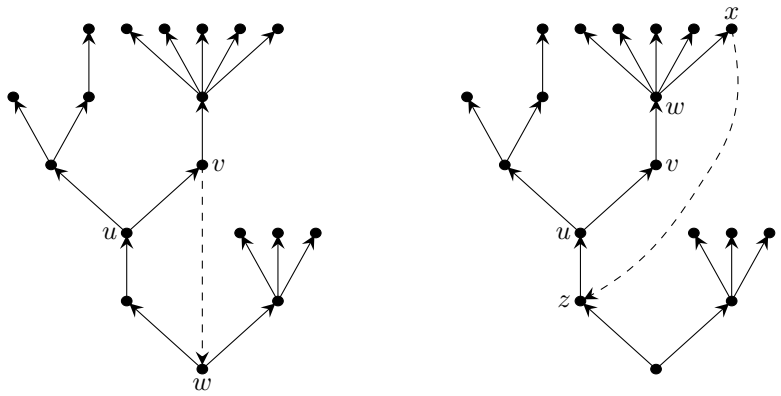

Figure 1.19: Finding a cycle that contains both uv and vw when (v, w) is not a leading edge.

(i) All tree edges incident with s are leading edges, and s is a cut vertex if and only if it is incident with at least two tree edges.

(ii) A vertex $v \in V(H) \setminus \{s\}$ is a cut vertex if and only if there is at least one leading edge (v, w).

Proof. (i) The first part of the statement follows directly from the definitions and the trivial fact that s has the smallest value of DFSnum in H. If the tree edges incident with s are $(s, w_1), \ldots, (s, w_k)$ (possibly $k = 0$), then $T[V(H)] - s$ consists of the components $T_{w_1}, \ldots, T_{w_k}$. By Proposition 1.4(iii), there are no edges of G between T_{w_i} and T_{w_j} for $i \neq j$ and thus, $H - s$ decomposes into components $H[V(T_{w_1})], \ldots, H[V(T_{w_k})]$. In particular, s is a cut vertex if and only if $k \geq 2$.

(ii) If (v, w) is a leading edge, then Lemma 1.16 implies that v lies in (at least) two blocks and thus is a cut vertex by Lemma 1.12(iv). For the reverse direction, suppose there is no such leading edge. Again by Lemma 1.16, each edge (v, x) of H lies in the same block as (u, v), where u is the parent of v. Furthermore, for each edge (y, v) with $y \neq u$, we have $y \in V(T_w)$ for some child w of v, which implies that yv and vw lie on a common cycle and thus in the same block by Lemma 1.12(iii). In other words, all edges incident with v lie in the same block. Hence v lies in only one block and thus, again by Lemma 1.12(iv), v is not a cut vertex. □

We can now use the information gathered so far to describe a linear-time algorithm that finds all blocks and cut vertices.

Theorem 1.18 (Tarjan 1972)**.** *There is an algorithm that determines the blocks and cut vertices of a graph G in linear time $\Theta(n + m)$.*

Proof. Run the DFSb algorithm (this takes time $\Theta(n + m)$). For roots of spanning trees of components of G, check which are cut vertices by Lemma 1.17(i) (time $O(n)$). Then, for each non-root vertex v, compare LowPoint[v] with DFSnum[parent[v]] (time $O(n)$) to identify the leading edges and remaining cut vertices (by Lemma 1.17(ii)).

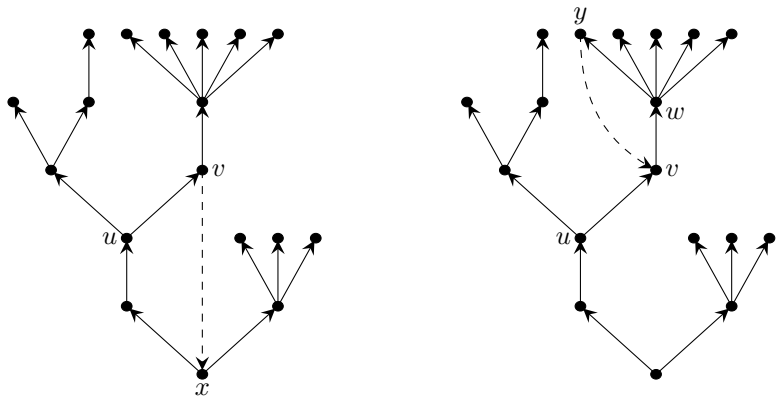

Figure 1.20: If there is no leading edge starting at v, then all edges of type (v, x) lie in the same block as the tree edge (u, v). All edges of type (y, v) lie in the same block as some tree edge (v, w), and thus also in the same block as (u, v).

It remains to determine the blocks. Consider the cut vertex v with the largest value DFSnum$[v]$. Let (v, w) be a leading edge, then there is no edge from T_w to $G - T_w - v$. Furthermore, $G[V(T_w) \cup \{v\}]$ has no cut vertex (because of the maximality of DFSnum$[v]$ among cut vertices) and thus is a block. Repeat this step for all leading edges (v, w). If v is the root of its component H, we have found all blocks of H and thus we can continue the procedure for $G - H$. Otherwise, delete each T_w as above from G and observe that v is then no longer a cut vertex. Now continue with the next cut vertex. Once there are no cut vertices left, the remaining graph consists only of components without cut vertices, all of which are therefore blocks. Again, we have running time $O(n)$. □

1.4 Edge-connectivity

Definition (k-edge-connectedness, edge-connectivity)**.** Let $k \in \mathbb{N}$. A graph G with at least two vertices is called *k-edge-connected* if, for every set $F \subset E(G)$ with $|F| < k$, the graph $G - F$ is connected.

We define the *edge-connectivity* of G as

$$\lambda(G) := \max\{k \in \mathbb{N} \colon G \text{ is } k\text{-edge-connected}\}.$$

Proposition 1.19. *For every graph G with at least two vertices, we have*

$$\kappa(G) \leq \lambda(G) \leq \delta(G).$$

Proof. Exercise. □

The bounds $\kappa(G)$ and $\delta(G)$ can be arbitrarily far apart and $\lambda(G)$ can lie at either end of this spectrum.

Theorem 1.20 (Edge version of Menger)**.** *For disjoint sets A, B of vertices in a graph G, the largest number of edge-disjoint A–B paths equals the smallest size of an edge set separating A and B.*

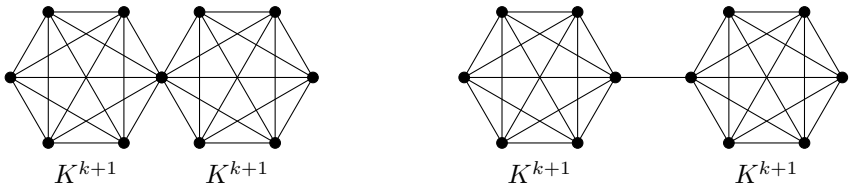

Figure 1.21: Two graphs with minimum degree k and connectivity 1. The graph on the left has edge-connectivity k, while the one on the right has edge-connectivity 1.

Proof. Exercise. □

Corollary 1.21. *For every $k \in \mathbb{N}$, a graph G with at least two vertices is k-connected if and only if any two distinct vertices are linked by k edge-disjoint paths.*

Definition (Edge-ear-decomposition)**.** An *edge-ear-decomposition* of a graph G is a sequence $(C, H_1, \dots, H_k)$ of subgraphs of G ($k = 0$ is allowed) such that

(i) C is a cycle;

(ii) each H_i is either a $(C \cup H_1 \cup \cdots \cup H_{i-1})$-path or a cycle that meets $C \cup H_1 \cup \cdots \cup H_{i-1}$ in precisely one vertex;

(iii) $G = C \cup H_1 \cup \cdots \cup H_k$.

Proposition 1.22. *A graph is 2-edge-connected if and only if it has an edge-ear-decomposition.*

Proof. Imitate the proof of Proposition 1.9. □

Chapter 2

Matchings

Definition (Matchings)**.** An independent set M of edges of a graph G is called *matching* (or *1-factor*). If uv lies in a matching M, we say that u *is matched to* v (and vice versa). In this case, we also say that u and v are *covered* by M. A matching is *perfect* if it covers all vertices of G. It is *almost perfect* if precisely one vertex is not covered.

2.1 Matchings in bipartite graphs

Definition (Vertex cover)**.** A *vertex cover* in a graph G is a set U of vertices such that each edge of G has an end vertex in U.

In a bipartite graph with sides A and B, matchings are precisely the sets of disjoint A–B paths, while vertex covers are the same as A–B separators. Thus, we have the following special case of Menger's theorem (which was originally proved independently by other methods).

Theorem 2.1 (König 1931)**.** *The largest size of a matching in a bipartite graph G equals the smallest size of a vertex cover.* □

Theorem 2.2 (Hall 1935; "Marriage theorem")**.** *A bipartite graph G contains a matching that covers all of A if and only if*

$$|N(S)| \geq |S|$$

holds for each $S \subseteq A$.[1]

Proof. The marriage condition is obviously necessary. Vice versa, suppose that the marriage condition holds and let U be a vertex cover of smallest size. By König's theorem, $|U|$ equals the size of a largest matching. We thus need to show that $|U| = |A|$. Every neighbour of a vertex in $A \setminus U$ lies in $B \cap U$, because G is bipartite and U is a vertex cover. In particular, $|N(A \setminus U)| \leq |B \cap U|$. By the marriage condition, applied to the set $A \setminus U$, we have

$$|A| = |A \cap U| + |A \setminus U| \leq |A \cap U| + |N(A \setminus U)| \leq |A \cap U| + |B \cap U| = |U| \,.$$

On the other hand, A is a vertex cover and hence $|A| \geq |U|$ by the minimality of U. Thus, $|U| = |A|$, as desired. □

[1] This is also called the *marriage condition*.

An alternative proof of Hall's theorem is more constructive and can be used to find a largest matching algorithmically.

Definition (Alternating path)**.** Given a matching M in a bipartite graph G, a path in G is called *alternating* if it starts at an unmatched vertex in A and alternately used edges from $E(G) \setminus M$ and M. An alternating path P is *augmenting* if it ends in an unmatched vertex in B. In that case, the symmetric difference $M \triangle P$ is a matching with $|M| + 1$ edges.

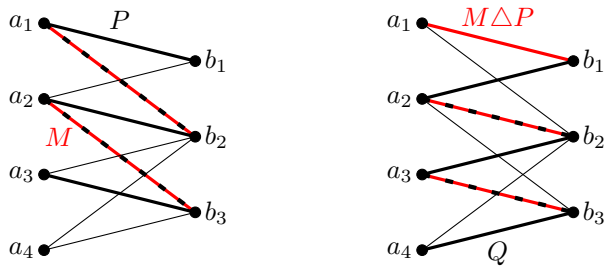

Figure 2.1: Augmenting and alternating paths in a bipartite graph. On the left, we have a matching M (red) and an augmenting path P (bold black) from a_3 to b_1. On the right, we have the matching $M \triangle P$ and a maximal alternating path Q (bold black), which is not augmenting.

Second proof of Theorem 2.2. Suppose that M is a matching consisting of less than $|A|$ edges; we shall prove that there is an augmenting path. Let A' and B' denote the sets of vertices in A and B, respectively, that lie on alternating paths. We have $N(A') \subseteq B'$, since for $ab \in E(G)$ with $a \in A'$, the alternating path ending in a either already contains b or can be extended to an alternating path by adding the edge ab. The marriage condition thus yields $|B'| \geq |A'|$.

For each $ab \in M$, either both vertices lie on an alternating path or neither of them do. Indeed, if an alternating path contains a, then a cannot be the first vertex in the path, and so the edge ab is in the path. Conversely, if an alternating path contains b and the path doesn't contain a, then b must be the final vertex on the path, and we can add the edge ab to form a longer alternating path.

Thus, there are the same number of matched vertices in A' as there are in B', and so since $|B'| \geq |A'|$, it follows that B' contains at least as many unmatched vertices as A' does. But A' contains all $|A| - |M| \geq 1$ unmatched vertices in A, because each such vertex forms an alternating path of length zero. Therefore, B' contains some unmatched vertex, which means that there exists an augmenting path. □

Proposition 2.3. *If M is a matching in a bipartite graph G that does not have largest size, then there is an augmenting path.*

Proof. Let M' be a matching with $|M'| > |M|$, where we may assume that M and M' are disjoint, otherwise we consider the graph $G - V(M \cap M')$ where $M' \setminus M$ is a larger matching that $M \setminus M'$. Let $A_{M'}$ and A_M be the sets of vertices in A that are covered by M' and M, respectively. Consider the graph

$H := (V(G), M \cup M')$ formed by the union of the two matchings. Since M and M' have maximum degree one, every component in H is a path or cycle, with edges alternating between M and M'. In particular, any component of $H := (V(G), M \cup M')$ that contains a vertex in $A_{M'} \setminus A_M$ is an alternating path P with respect to M. If P ends in a vertex $b \in B$, then b is isolated in M and P is augmenting. Hence, if P is not augmenting, then it has even length and so ends in a vertex in $A_M \setminus A_{M'}$.

Since $|M'| > |M|$ it follows that $|A_{M'}| > |A_M|$ and so $|A_{M'} \setminus A_M| > |A_M \setminus A_{M'}|$. In particular, since each component contains at most one vertex of $A_{M'} \setminus A_M$, there must be some component which contains a vertex of $A_{M'} \setminus A_M$ and no vertex of $A_M \setminus A_{M'}$, and hence is an augmenting path. □

We can utilise Proposition 2.3 to create an algorithm that finds a largest matching by starting with an empty matching and then recursively increasing the current matching M via an augmenting path, which we find as follows. Suppose that M covers the vertices in $A_M \subseteq A$ and in $B_M \subseteq B$. In order to either find an augmenting path or prove that none exist, we use a modified BFS that starts with a queue Q listing all vertices in $A \setminus A_M$. The BFS then alternately uses non-matching edges and matching edges and stops once unmatched vertices in B are found or the whole graph has been discovered.

For $b \in B_M$, we denote by $a[b]$ the vertex in A_M that b is matched to. Any unmatched vertices in B found by the algorithm will be collected in a set B'.

```
PROCEDURE BFSm(G,Q)
BEGIN
   B' := ∅;
   WHILE (B' = ∅ AND Q ≠ ∅){
      v := First(Q);
      IF (v ∈ A){
         FOR (w ∈ N(v)){
            IF (known[w] = FALSE){
               known[w] := TRUE;
               parent[w] := v;
               Concatenate (Q,w);
               IF (w ∉ B_M) THEN B' := B' ∪ {w};
            }
         }
      }
      ELSE{
         known[a[v]] := TRUE;
         parent[a[v]] := v;
         Concatenate (Q,a[v]);
      }
      Remove v from Q;
   }
END
```

Observe that if the vertex v considered in the WHILE-loop lies in A, then BFSm discovers all neighbours of v that have not been discovered before. If this includes unmatched vertices, the WHILE-loop will stop after it is finished dealing with v. Thus, a vertex in B will only be considered as v if it is matched. In that case, the algorithm will add the vertex $a[v]$ next.

If BFSm finds no unmatched vertices in B (i.e. $B' = \emptyset$), then there are no augmenting paths with respect to M (exercise). Vice versa, if we have found some $b \in B'$, then the path consisting of b, its parent, the parent of its parent, and so on, is an augmenting path (in reverse direction).

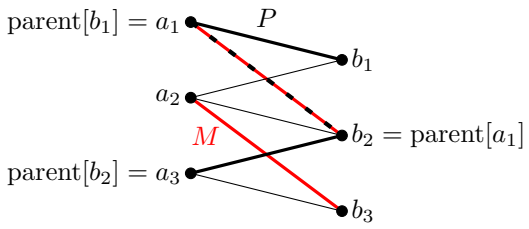

Figure 2.2: The augmenting path P belonging to the unmatched (in M) vertex b_1 found by BFSm.

The following procedure determines the symmetric difference of M and this augmenting path ending in b.

```
PROCEDURE AUGMENT(B_M,a[·],b)
BEGIN
    B_M := B_M ∪ {b};
    a[b] := parent[b];
    WHILE (a[b] ∈ A_M){
        b := parent[a[b]];
        a[b] := parent[b];
    }
    A_M := A_M ∪ {a[b]};
END
```

The main program sets the initial values and recursively increases the matching until we have found a largest matching.

```
BEGIN
    A_M, B_M := ∅;
    Queue Q;
    augm := TRUE;
    WHILE (augm = TRUE){
        FOR (v ∈ A_M ∪ B) DO known[v] := FALSE;
        Q := ∅;
        FOR (v ∈ A \ A_M){
            known[v] := TRUE;
            Concatenate (Q,v);
        }
        BFSm(G,Q);
        IF (B' ≠ ∅){
            Pick b ∈ B';
            AUGMENT(B_M,a[·],b);
        }
        ELSE augm := FALSE;
    }
END
```

Hopcroft-Karp algorithm. The above algorithm is reasonably fast (exercise), but not as fast as possible. We give a sketch of a faster algorithm found by Hopcroft and Karp in 1973. Again, start with $M = \emptyset$.

(i) When BFS encounters the first unmatched vertex in B, the algorithm finishes to fill the current layer (unlike BFSm, which only finished dealing with the current vertex). This way, several unmatched vertices in B are found, let B' be their set.

(ii) Instead of taking a single augmenting path, the algorithm finds a maximal set (maximal with respect to inclusion, not necessarily of maximum size!) of disjoint augmenting paths by applying a DFS to the vertices in B' going backwards through the layers. Observe that in our algorithm above, we used a fixed augmenting path for each vertex in B'; using DFS instead allows for other augmenting paths that we would not be able to use.

(iii) Take the symmetric difference of M and these augmenting paths.

(iv) Repeat.

It is known (but not obvious) that under the assumption $m \geq n$, the algorithm needs only $O(\sqrt{n})$ rounds, each of which takes time $O(m)$. We thus have total running time $O(\sqrt{n}m)$.

2.2 Stable matchings

In many applications, every vertex in the graph comes with a list of preferences for its neighbours and the goal is to match every vertex to a neighbour that is as high as possible on its list.

Definition (Stable matching)**.** Let G be a graph and suppose that each vertex $v \in V(G)$ has a total order $\leq_v$ on the set $N(v)$ of its neighbours. For neighbours u, w of v, we say that v *prefers* u over w if $u <_v w$ (i.e. "smaller means better").

We call a matching M in G *stable* if there is no edge $ab \in E(G) \setminus M$ such that both a and b are either unmatched or prefer each other over the vertex they are matched to in M.[2]

Observe that every stable matching M is maximal with respect to inclusion, because if $M' \supsetneq M$ is a matching, then any edge in $M' \setminus M$ is a witness for M *not* being stable. However, stable matchings are not necessarily largest matchings.

Moreover, it is easy to construct graphs (together with lists of preferences) that do not have stable matchings at all.

Again, the situation becomes easier if we consider bipartite graphs.

Theorem 2.4. *Every bipartite graph with preferences has a stable matching.*

In order to prove Theorem 2.4, let us construct an algorithm that finds a stable matching in any given bipartite graph G. We are going to use a system of "proposing" and "rejecting". In the beginning, each vertex $a \in A$ would

[2]The idea behind this notation is that if M is not stable and ab is an edge witnessing this fact, then a and b would "break up" with their respective partners and rather create a new edge together.

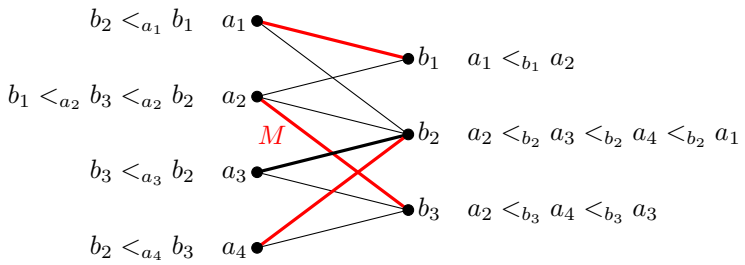

Figure 2.3: Stable and non-stable matchings in a bipartite graph with preferences. The matching M (red) is not stable, as witnessed by the edge a_3b_2. The matching $(M \setminus \{a_4b_2\}) \cup \{a_3b_2\}$ is stable (check!).

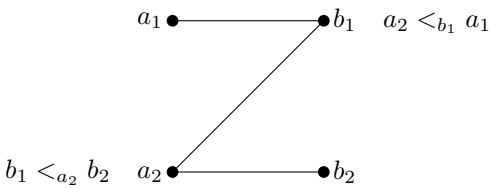

Figure 2.4: A bipartite graph in which the unique largest matching $\{a_1b_1, a_2b_2\}$ is not stable. The only stable matching in this graph is $\{a_2b_1\}$.

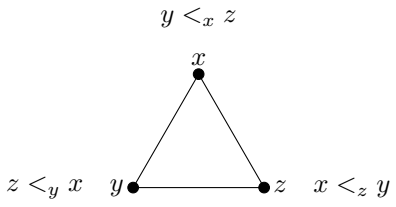

Figure 2.5: A graph with no stable matching.

prefer to be matched to its neighbour that comes first in the order $<_a$, so let each $a \in A$ propose to that neighbour. If each vertex in B receives at most one proposal, then we could match each vertex in A to its preferred partner and obtain a matching that would automatically be stable, because no vertex in A would prefer to "break up" with its partner.

Otherwise, if $b \in B$ receives more than one proposal, then it will prefer the one among the proposing vertices that comes earliest in the order $<_b$, so let b reject all other proposals. Then all vertices in A make new proposals (the ones that were rejected propose to the next best neighbour on their list, the others propose to the same neighbour as in the previous step), the vertices in B reject some of them and so on. Observe that a proposal from a to b that is not rejected in the first round, say, might be rejected in the second round, because some vertex $a' <_b a$ might now propose to b. If no proposal is rejected in some round, we take the currently open proposals as our matching. (Observe that it is not immediate that this is a stable matching.)

Let us formalise this algorithm. For the sake of notation, suppose that the preferences are encoded by a queue $N[v]$ for each vertex v, with the entry $\emptyset$ being the last element (i.e. each vertex prefers *any* possible partner over not being matched at all).

```
PROCEDURE STABLE(G)
BEGIN
   FOR (b ∈ B) DO proposal[b] := ∅;
   reject := TRUE;
   WHILE (reject = TRUE){
      reject := FALSE;
      FOR (a ∈ A) DO PROPOSE(G,a);
   }
END
```

```
PROCEDURE PROPOSE(G,a)
BEGIN
   b := First(N[a]);
   IF (b ≠ ∅){
      IF (a <_b proposal[b]){
         IF (proposal[b] ≠ ∅){
            Remove b from N[proposal[b]];
            reject := TRUE;
         }
         proposal[b] := a;
      }
      ELSE IF (a >_b proposal[b]){
         Remove b from N[a];
         reject:= TRUE;
      }
   }
END
```

Note that in terms of the output, it does not matter whether the algorithm first gathers all proposals from vertices in A and then proceeds with rejecting some of them, or (like STABLE does) immediately rejects any proposal that is currently not the favourite choice for the vertex being proposed to.

Proposition 2.5. *For every bipartite graph G with preferences, the set*

$$M := \big\{\{b, \text{proposal}[b]\} : b \in B \text{ with } \text{proposal}[b] \neq \emptyset\big\}$$

produced by STABLE(G) is a stable matching.

Proof. By construction, no vertex in B is incident with more than one edge of M. At every time of the algorithm, every vertex $a \in A$ has at most one open proposal (that is, there is at most one $b \in B$ with $a = \text{proposal}[b]$). Thus, M is a matching.

Let $ab \in E(G) \setminus M$. If a never proposed to b, then some proposal of a to a vertex $b' <_a b$ was never rejected and thus $ab' \in M$. If a proposed to b at some point, then this proposal was eventually rejected and thus $a'b \in M$ for some vertex $a' <_b a$. As this holds for all choices of ab, M is stable. □

Proposition 2.5 in particular implies Theorem 2.4.

Remark. *Every vertex $b \in B$ rejects at most $d(b) - 1$ proposals. By construction, the WHILE-loop in STABLE stops after the first time that one of its runs features no rejection. Thus, the loop runs at most*

$$1 + \sum_{b \in B} (d(b) - 1) = m - |B| + 1$$

times and each run takes time $\Theta(|A|)$. Thus, we have a total running time of

$$O\Big(\, |A| \,\big(m - |B| + 1\big) + |B| \,\Big).$$

(Setting of the initial conditions takes time $\Theta(|B|)$.)

We now know that stable matchings always exist in bipartite graphs. But what can we say about their structure? Let us compare two stable matchings.

Proposition 2.6. *Let M, M' be stable matchings in a bipartite graph G. Then the components of the graph $H := (V(G), M \cup M')$ are*

- *vertices unmatched in both M and M',*
- *edges from $M \cap M'$ (plus their end vertices), and*
- *even cycles consisting alternately of edges from M and M' such that each vertex in the cycle prefers its successor in the cycle over its predecessor.*

Proof. Each vertex in H has degree at most two, hence its components are paths (possibly of length zero) and cycles. Let C be any component of H.

If C is a path of length zero, then its vertex is unmatched in both M and M'. If C is a path of length one, then its edge e lies in one of the matchings, say in M. Since e forms a component of H, either $e \in M'$ or both its end vertices are unmatched in M'. But the latter would be a contradiction to M' being stable, which means that $e \in M \cap M'$.

Suppose that C is a path $v_0 \dots v_k$ of length at least two and that $v_0v_1 \in M$. This means that $v_1v_2 \in M'$, $v_2v_3 \in M$, and so on, because M, M' are matchings. Every v_i with $1 \leq i \leq k-1$ either prefers v_{i-1} or v_{i+1} over the other. The vertex v_1 has to prefer v_2, because otherwise M' would not be stable (as witnessed by

v_0v_1). Let $i \in \{1, \dots, k-1\}$ be the largest index for which v_i prefers v_{i+1} over v_{i-1}. Let $\tilde{M} \in \{M, M'\}$ be the matching that does *not* contain $e = v_iv_{i+1}$. Then e shows that $\tilde{M}$ is not stable, because v_i prefers v_{i+1} over v_{i-1} (to which v_i is matched in $\tilde{M}$), while v_{i+1} either is not matched in $\tilde{M}$ (if $i = k-1$) or it prefers v_i over v_{i+2} (to which v_{i+1} is matched in $\tilde{M}$ if $i < k-1$) by the maximality of i. Since this is a contradiction, C cannot be a path of length at least two.

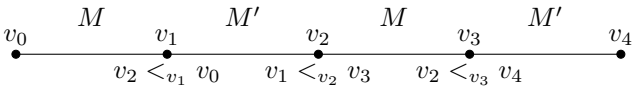

Figure 2.6: A path of length at least two consisting of edges from two matchings M, M'. The edge v_1v_2 shows that M is not stable.

Finally, suppose that C is a cycle. Then C has even length (because G is bipartite) and consists alternately of edges from M and M' (because these are matchings). Suppose that some vertices on C prefer their predecessor and some other vertices prefer their successor. Then there would be consecutive vertices u, v on the cycle such that u prefers v and vice versa. If $uv \in M$, then uv is a witness that M' is *not* stable, a contradiction. Thus, we can choose the direction of the cycle in such a way that each vertex prefers its successor. □

As an immediate corollary, we deduce that all stable matchings in a bipartite graph cover the same vertices, a result that looks rather surprising.

Corollary 2.7. *If G is bipartite, then there exist sets $A' \subseteq A$ and $B' \subseteq B$ such that* every *stable matching in G is a perfect matching of $G[A' \cup B']$.* □

Remark. *The stable matching M found by* STABLE *is* best possible *for each vertex $a \in A'$ in the sense that if a is matched to b, then every other stable matching matches a to some b' with $b \leq_a b'$ (exercise). For the vertices in B', however, M is* worst possible *in the same sense.*

2.3 Matchings in general graphs

Suppose that a graph G has a perfect matching M, what can we say about the structure of G? Arguably, the most obvious property of G is that it has an even number of vertices. We also know that each component of G is even.

A slightly less obvious fact becomes apparent when we consider $G - S$ for some $S \subseteq V(G)$. Clearly, M can contain at most $|S|$ edges between S and $V(G - S)$. Thus, M induces on $G - S$ a matching that covers all but at most $|S|$ vertices, which implies that $G - S$ has at most $|S|$ odd components.

Surprisingly, this condition is enough to guarantee a perfect matching.

Theorem 2.8 (Tutte 1947)**.** *A graph G has a perfect matching if and only if*

$$\forall S \subseteq V(G) \colon G - S \text{ has at most } |S| \text{ odd components.} \tag{2.1}$$

We also call (2.1) the *Tutte condition*.

We shall prove Theorem 2.8 via a more general result by Gallai and Edmonds. Before we do that, let us note that Tutte's theorem yields a very short proof of a result that was originally proved with significantly more effort.

Theorem 2.9 (Petersen 1891)**.** *Every bridgeless cubic graph has a perfect matching.*

Proof. Let G be bridgeless and cubic and let $S \subseteq V(G)$ be given. For any $U \subseteq V(G-S)$, denote by $E(U,S)$ the set of edges between U and S. Let H be a component of $G-S$. We have

$$3\,|H| = \sum_{v \in V(H)} d_G(v) = 2\,\|H\| + |E(V(H),S)|\,,$$

because every edge of H is counted twice in the sum, while every edge between $V(H)$ and S is counted once. Thus, if H has an odd number of vertices, then $|E(V(H),S)|$ is odd as well. As G is bridgeless, this means that there are at least three edges between $V(H)$ and S. The total number of edges between S and $V(G)-S$ is bounded by $3\,|S|$. Therefore, there can be at most $|S|$ many odd components. Thus, G satisfies the Tutte condition and hence has a perfect matching by Theorem 2.8. □

If G has a perfect matching M and for some set S we have precisely $|S|$ odd components in $G-S$, then M has to contain, for each such odd component H,

- precisely one edge between S and $V(H)$,
- an almost perfect matching of H.

Inspired by this observation, let us define the following notation.

Definition (Factor-critical graphs, matchable sets)**.** Call a non-empty graph G *factor-critical* if, for each $v \in V(G)$, the graph $G-v$ has a perfect matching.

Let a graph G and a set $S \subseteq V(G)$ of vertices be given and suppose that $G-S$ has components $H_1,\dots,H_k$. We call S *matchable to* $G-S$ if the graph $G' := G/(V(H_1),\dots,V(H_k)) - E(G[S])$ contains a matching covering all of S.

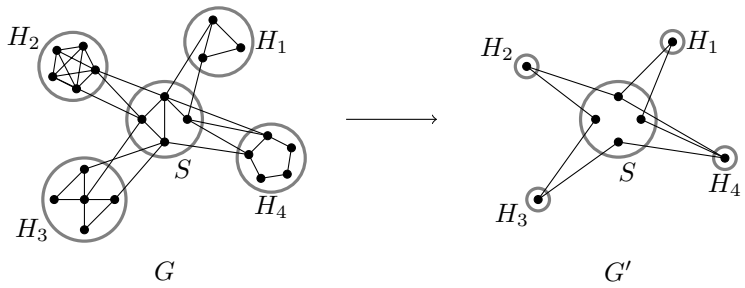

Figure 2.7: How to construct the graph G' from G. In this example, all components of $G-S$ are factor-critical and S is matchable to $G-S$ (check!).

Clearly, factor-critical graphs are always odd. Observe that unless $S = \emptyset$ or $S = V(G)$, the graph G' is bipartite with sides S and $\{V(H_1),\dots,V(H_k)\}$. The set $S = \emptyset$ is always matchable to $G-S$.

Theorem 2.10 (Gallai-Edmonds 1964/65)**.** *Every graph G contains a vertex set S such that*

(i) each component of $G - S$ is factor-critical and

(ii) S is matchable to $G - S$.

If S is any such set, then G has a perfect matching if and only if $G - S$ has precisely $|S|$ components.

Proof of Theorem 2.8 from Theorem 2.10. The Tutte condition is clearly necessary for a perfect matching. Vice versa, suppose that the Tutte condition holds and let S be the set provided by Theorem 2.10. Then $G - S$ has at least $|S|$ components by (ii), and each component is odd by (i). Together with the Tutte condition for the set S, this implies that $G - S$ has precisely $|S|$ components. Thus, G has a perfect matching by the last statement of Theorem 2.10. □

Observe that we have actually not only proved that Theorem 2.10 implies Theorem 2.8, but the following stronger statement.

> If the statement of Theorem 2.10 is true for a graph G, then the statement of Theorem 2.8 also holds for G.

Proof of Theorem 2.10. We shall show the existence of S by induction on $|G|$. For empty G, we can choose the empty set as S. For the induction step, suppose that G is non-empty and that Theorem 2.10 (and thus also Theorem 2.8) is true for all graphs on fewer vertices. For every graph H, denote by $q(H)$ the number of odd components of H.

For all vertex sets $T \subseteq V(G)$, set

$$f(T) := q(G - T) - |T|\,.$$

In other words, $f(T) \leq 0$ is equivalent to the Tutte condition for T, while $f(T) > 0$ is a measure of how bad the Tutte condition fails for T. Among all sets T with greatest value $f(T)$, choose S to be a largest one. Then

$$f(S) \geq f(\emptyset) \geq 0.$$

We will show that S has the desired properties.

To prove (i), suppose, for contraction, that for some vertex v in a component H of $G - S$, the graph $H - v$ has no perfect matching. As $H - v$ is smaller than G, we know that Theorem 2.8 holds for $H - v$ by the induction hypothesis. Thus, there is a set $S' \subseteq V(H - v)$ for which the Tutte condition fails, that is, $q(H - v - S') \geq |S'| + 1$. This means that

$$\begin{aligned} f(S \cup \{v\} \cup S') &= q(G - S - v - S') - |S \cup \{v\} \cup S'| \\ &= \big(q(G - S) - q(H) + q(H - v - S')\big) - \big(\,|S| + 1 + |S'|\,\big) \\ &= f(S) + q(H - v - S') - |S'| - \begin{cases} 2 & \text{if } H \text{ is odd,} \\ 1 & \text{if } H \text{ is even.} \end{cases} \end{aligned}$$

If H is even, recall that $q(H - v - S') \geq |S'| + 1$, and so $f(S \cup \{v\} \cup S') \geq f(S)$, contradicting our choice of S.

If H is odd, then

$$|H - v| = |H - v - S'| + |S'|$$

is even and thus $|H - v - S'|$ and $|S'|$ are either both even or both odd. Since $|H - v - S'|$ is even if and only if $q(H - v - S')$ is even, $q(H - v - S')$ and $|S'|$ have the same parity, implying that $q(H - v - S') \geq |S'| + 2$. Either way, we have $f(S \cup \{v\} \cup S') \geq f(S)$, contradicting the choice of S. This proves (i). In particular, we may assume that every component of $G - S$ is factor-critical, and hence odd.

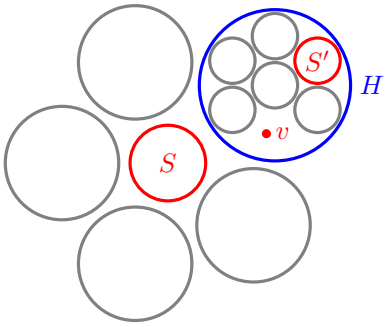

Figure 2.8: If $H - v$ does not have a perfect matching, we find a set S' that violates the Tutte condition in H. The odd components of $G - (S \cup \{v\} \cup S')$ are precisely the odd components of $G - S$ apart from H, if H is odd, (these are $q(G - S) - q(H)$ odd components) and the $q(H - v - S')$ odd components of $H - v - S'$.

Now suppose, for contradiction, that (ii) fails. Then in particular $S \neq \emptyset$. We also have $S \neq V(G)$, because $f(V(G)) = -|G| < 0 \leq f(S)$. Thus, the graph G' obtained from G by contracting all components of $G - S$ and deleting all edges in $G[S]$ is bipartite with S being one side. Now the fact that (ii) fails implies by Hall's theorem (Theorem 2.2) that there is a set $S'' \subseteq S$ for which the marriage condition in G' fails, that is, S'' sends edges to less than $|S''|$ components of $G - S$. In particular, $S'' \neq \emptyset$. But then all components of $G - S$ that do not send an edge to S'' are also components of $G - (S \setminus S'')$, meaning that, since all components of $G - S$ are odd, $q(G - (S \setminus S'')) > q(G - S) - |S''|$ and thus

$$f(S \setminus S'') = q(G - (S \setminus S'')) - |S \setminus S''| > q(G - S) - |S| = f(S),$$

a contradiction.

It remains to prove the last statement of Theorem 2.10. To that end, let S be *any* set satisfying (i) and (ii). Then by (ii), $G - S$ has at least $|S|$ components, all of which are odd by (i). If $G - S$ has more than $|S|$ (odd) components, then G does not have a perfect matching by the trivial direction of Tutte's theorem. If $G - S$ has precisely $|S|$ components, take a matching that matches each vertex in S to a vertex in a different component of $G - S$ (such a matching exists by (ii)). For each component of $G - S$, exactly one vertex is now matched and thus we can find a perfect matching in the rest of the component by (i). Together, this is a perfect matching of G. □

Suppose that S is the set provided by Theorem 2.10 and denote again by $q(G - S)$ the number of odd components of $G - S$. Then by Theorem 2.10, there is a matching that consists of $|S|$ edges between S and distinct components of

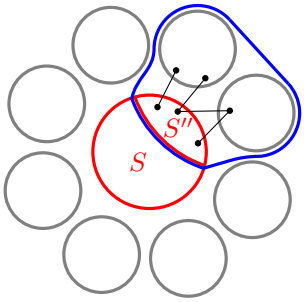

Figure 2.9: If S'' is witnessing the fact that S is not matchable to $G - S$, then the odd components of $G - (S \setminus S'')$ are at least the odd components of $G - S$ that have no neighbour in S''. The components of $G - (S \setminus S'')$ contained in S'' and its neighbouring components of $G - S$ (blue) might or might not be odd.

$G - S$ and an almost perfect matching of each component of $G - S$, thus covering

$$2\,|S| + |G - S| - q(G - S)$$

vertices.

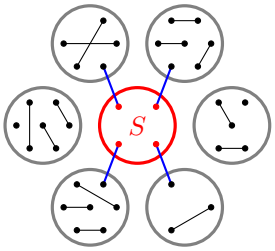

Figure 2.10: Each largest matching consists of $|S|$ edges from S to distinct components of $G - S$ (blue) and an almost perfect matching in each component of $G - S$ (black).

If M is *any* matching, then it can cover at most $|G - S| - q(G - S)$ vertices by edges within $G - S$, and if it does cover this many vertices, then it induces an almost perfect matching of each component of $G - S$. Furthermore, by edges not in $G - S$, M can cover at most $2\,|S|$ vertices, and equality holds if and only if M matches each vertex in S to a vertex in $G - S$. We thus have the following structural statement.

Corollary 2.11. *Let G be a graph and $S \subseteq V(G)$ be a set provided by Theorem 2.10. Then every matching in G of largest size consists of $|S|$ edges between S and $G - S$ and an almost perfect matching of each component of $G - S$.* □

Chapter 3

Planar graphs

In this chapter we shall frequently use seemingly obvious, but actually non-trivial topological facts, such as the Jordan curve theorem that any simply closed curve separates the plane into precisely two components. We will, however, point out the places where we use such facts.

3.1 Basic properties of planar graphs

Definition (Plane and planar graphs, faces)**.** A *plane graph* is a drawing of a graph in the plane such that edges only intersect in their common end vertices (if they have any in common). Formally,

- we assign to each vertex of the graph a different point in the plane;
- each edge is represented by a homeomorphic image of the unit interval $[0, 1]$ between its end vertices;
- edges meet if and only if they share an end vertex. In that case, they meet *only* in said vertex.

A *planar graph* is a graph that has a drawing.

If G is a plane graph, then we call the components of $\mathbb{R}^2 \setminus G$ the *faces* of G. The unique unbounded face is the *outer face*. Every face of G is an open set. By the *boundary* of a face we mean the subgraph of G formed by the vertices and edges that are contained in the (topological) closure of the face.

Clearly, planarity is closed under taking subgraphs. It is also easy to see that a graph is planar if and only if its components are planar. Slightly less obvious is that the same also holds for blocks instead of components.

Proposition 3.1. *A graph is planar if and only if all its blocks are planar.*

Proof. The "only if" direction is trivial. For the opposite direction, let G be a graph all of whose blocks are planar. By the fact that the block-cut-vertex-graph is a forest (Proposition 1.13), we can enumerate the blocks of G as $B_1, \dots, B_k$ so that each B_i meets at most one vertex of $B_1 \cup \cdots \cup B_{i-1}$. We shall construct a a drawing of G by induction on k.

For $k \leq 1$, planarity of G is trivial. Assume that $k \geq 2$ and that $H := B_1 \cup \cdots \cup B_{k-1}$ is planar. Fix a drawing of H. If B_k is disjoint from H, then we can just pick a drawing of B_k and add it to the drawing of H in a disjoint way. Otherwise, let v be the cut vertex in which B_k and H intersect and let F be a face of H whose boundary contains v. Choose a drawing of B_k that has v on the boundary of the outer face.[1] This drawing of B_k can (with a little stretching) be inserted into F so that the two appearances of v coincide. The result is a drawing of G. □

Proposition 3.2 (Euler's formula)**.** *Every connected plane graph G has precisely*

$$\|G\| - |G| + 2$$

faces. If G is not connected, it has more than $\|G\| - |G| + 2$ faces.

A formal proof of Euler's formula needs a couple of topological tools. The basic proof idea is to use induction on $\|G\| - |G|$, with the class of trees forming the induction base. (Trees have $|G| - 1$ edges by Theorem 1.1 and a drawing of a tree has only one face,[2] hence they satisfy Euler's formula.) For a connected graph that is not a tree, we can delete an edge that does not disconnect the graph (again by Theorem 1.1), thus decreasing the number of faces by one.[3] Then we use the induction hypothesis on the smaller graph.

Proposition 3.3. *Suppose that G is a planar graph which is not a forest. If G contains no cycles of length shorter than $r \geq 3$, then*

$$\|G\| \leq \frac{r}{r-2}(|G| - 2).$$

In particular, every planar graph G on $n \geq 3$ vertices has at most $3n - 6$ edges.

Proof. Fix a drawing of G and let $\mathcal{F}$ be the set of faces of this drawing. For each $F \in \mathcal{F}$, denote by $e(F)$ the number of edges on its boundary. Each such boundary contains a cycle[4] and thus $e(F) \geq r$. Since each edge lies in at most two face boundaries, we have

$$r\,|\mathcal{F}| \leq \sum_{F \in \mathcal{F}} e(F) \leq 2\,\|G\|\,.$$

(In some situations, it is common that if an edge lies in only one face boundary, then it is counted twice for that boundary, thus yielding an equality on the right hand side above.) By Euler's formula, this implies

$$\frac{2}{r}\,\|G\| \geq |\mathcal{F}| \geq \|G\| - |G| + 2 \quad \Longrightarrow \quad \|G\| \leq \frac{r}{r-2}(|G| - 2).$$

The last claim follows immediately by the trivial observation that no cycle can have length shorter than three, and the fact that forests on $n \geq 3$ vertices have at most $n - 1 < 3n - 6$ edges by Theorem 1.1. □

[1] This can for instance be achieved by starting with any drawing, transfer it to a drawing on the sphere by applying stereographic projection (in the reverse direction), pick a point p in a face whose boundary contains v, and then apply stereographic projection again with projection point p.

[2] Which is non-trivial to prove rigorously, as it needs some kind of inverse Jordan curve theorem.

[3] Which is also not trivial.

[4] Not trivial.

Observe that the general upper bound in Proposition 3.3 also holds for forests, as long as $r \leq 2n - 2$.

Definition (Maximally plane/planar graphs, triangulations)**.** We call a plane graph *maximally plane* if we cannot add an edge without crossing an already existing edge. A planar graph is *maximally planar* if adding an edge between any two non-adjacent vertices results in a non-planar graph.

A *triangulation* is a plane graph in which all face boundaries are triangles.

Suppose that we draw a planar graph G and find that this drawing is maximally plane. Does this automatically tell us that G is maximally planar? Or might there be a different drawing of G that is not maximally plane? The next result tells us that the decision whether G is maximally planar does in fact *not* depend on how we draw it.

Proposition 3.4. *The following statements are equivalent for every plane graph G on $n \geq 3$ vertices.*

(i) *G is maximally planar;*

(ii) *G is maximally plane;*

(iii) *G is a triangulation;*

(iv) *$\|G\| = 3n - 6$.*

Proof. Exercise. □

3.2 Kuratowski's theorem

Non-planarity of a graph is usually very hard to prove from scratch. In this regard, Proposition 3.3 provides a helpful criterion. For instance, we know that the complete graph K^5 is not planar, because it has $10 > 9 = 3 \cdot 5 - 6$ edges. The complete bipartite graph $K_{3,3}$ is also not planar. To see this, observe that if a graph G is bipartite, it does not contain any triangles and thus, Proposition 3.3 is applicable with $r = 4$, giving an upper bound of $2|G| - 4$ edges. But $K_{3,3}$ has $9 > 8 = 2 \cdot 6 - 4$ edges.[5] Observe that the bound $3n - 6$ is *not* enough to prove that $K_{3,3}$ is not planar.

Notation. Let G, H be graphs. We write

- $G = TH$ if G is a subdivision of H;
- $TH \subseteq G$ if H is a topological minor of G;
- $G = MH$ if H is obtained from G by contracting connected vertex sets;
- $MH \subseteq G$ if H is a minor of G.

Remark. *If $G = TH$, then also $G = MH$. This implies that if H is a topological minor of G, then it is also a (regular) minor.*

Moreover, if G is planar, then all its (topological) minors are also planar. In particular, no planar graph can contain K^5 or $K_{3,3}$ as a (topological) minor.

[5] For both K^5 and $K_{3,3}$, showing non-planarity directly is also possible, but needs the same type of topological statements like we used in our proof sketch of Euler's formula.

Surprisingly, this simple condition of not containing K^5 and $K_{3,3}$ as (topological) minors is also sufficient for planarity.

Theorem 3.5 (Kuratowski 1930)**.** *A graph is planar if and only if it contains neither K^5 nor $K_{3,3}$ as a topological minor.*

Theorem 3.6 (Wagner 1937)**.** *A graph is planar if and only if it contains neither K^5 nor $K_{3,3}$ as a minor.*

We first note that since every (topological) minor of a planar graph is planar, and we have already shown that K^5 and $K_{3,3}$ are non-planar, the forward implication of both theorems is trivial.

Furthermore, it is clear that Kuratowski's theorem implies Wagner's theorem. Indeed, if G is planar, then by Kuratowski's theorem we have $MK^5, MK_{3,3} \not\subseteq G$ and so in particular $TK^5, TK_{3,3} \not\subseteq G$.

In fact, we will show that it is also relatively easy to deduce Kuratowski's theorem from Wagner's theorem, and hence the two theorems are equivalent.

In order to do so we will use the following useful lemma about the number of leaves in a tree, whose proof is left as an exercise.

Lemma 3.7. *Let T be a tree and let $V_i = \{x \in V(T) \colon d(x) = i\}$ be the set of vertices of degree i. Then*

$$|V_1| = 2 + \sum_{i \geq 2} (i-2)|V_i|.$$

Proof. Exercise. □

Proposition 3.8. *Theorems 3.5 and 3.6 are equivalent.*

Proof. We have already seen that Kuratowski's theorem implies Wagner's theorem; it remains to show the reverse implication. To that end, it will suffice to show that $TK^5, TK_{3,3} \not\subseteq G$ implies $MK^5, MK_{3,3} \not\subseteq G$. We shall prove the equivalent statement that each graph which contains an MK^5 or an $MK_{3,3}$ also contains a TK^5 or a $TK_{3,3}$.

Suppose first that $MK_{3,3} \subseteq G$. Let $H \subseteq G$ be minimal with $H = MK_{3,3}$ and denote by $V_1, \ldots, V_6$ the connected vertex sets with $K_{3,3} = H/(V_1, \ldots, V_6)$. By the minimality of H, each graph $H[V_i]$ is a tree and sends exactly one edge each to three other sets V_j. Thus, the graph consisting of all vertices in $V_i \cup N(V_i)$, together with all edges that are incident with at least one vertex in V_i, is a tree, denote it by T_i. The leaves of T_i are precisely its three vertices outside V_i, because any leaf in V_i could be deleted from H, contradicting the minimality of H. Note that, by Lemma 3.7 any tree with exactly three leaves has one vertex of degree three, while all other vertices have degrees two or one.

In particular, if we let v_i be the unique vertex of degree three in T_i, it follows that H is a subdivision of $K_{3,3}$ with branch vertices $v_1, \ldots, v_6$.

Now suppose that $MK^5 \subseteq G$ and let $H \subseteq G$ be minimal with $H = MK^5$. Denote $V_1, \ldots, V_5$ and $T_1, \ldots, T_5$ as before. (This time, each T_i has four leaves, one in each V_j with $j \neq i$.) Note that, by Lemma 3.7, every tree with four leaves contains either one vertex of degree four, or two vertices of degree three, with all other vertices of degree two or one

If each T_i has a vertex v_i of degree four, then as before H is a subdivision of K^5 with branch vertices $v_1, \ldots, v_5$.

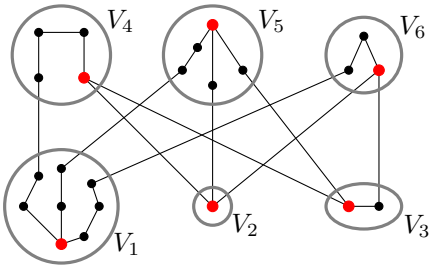

Figure 3.1: Finding a $TK_{3,3}$ in an $MK_{3,3}$. The vertices $v_1, \ldots, v_6$ in $V_1, \ldots, V_6$, respectively, are drawn in red.

We may thus assume that one of the trees, without loss of generality T_1, has no vertex of degree four. Then V_1 contains two vertices u_1, w_1 of degree three in T_1. Partition V_1 into two connected sets U_1 and W_1 that contain u_1 and w_1, respectively. Note that, since $H[V_1]$ is a tree, there is a unique edge from U_1 to W_1 and so, by the same arguments as in the previous case, $H[U_1 \cup N(U_1)]$ and $H[W_1 \cup N(W_1)]$ are trees with a unique vertex of degree three and all other vertices of degree two, whose three leaves are $N(U_1)$ and $N(W_1)$, respectively. Note that, then, $N(U_1)$ meets two V_i with $i \geq 2$ and $N(W_2)$ meets the other two.

Then

$$H' := H/(U_1, W_1, V_2, \ldots, V_5)$$

contains a $K_{3,3}$, because if U_1 has a neighbour in V_2 and V_3, say, then each of U_1, V_4, V_5 is adjacent to each of W_1, V_2, V_3. Therefore, $MK_{3,3} \subseteq G$ and thus also $TK_{3,3} \subseteq G$ by the first part of the proof. □

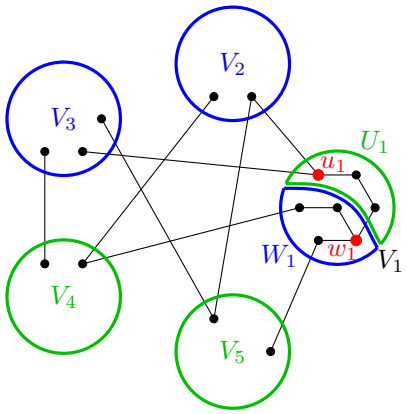

Figure 3.2: Finding an $MK_{3,3}$ in an MK^5. Note that the MK^5 also contains and edge between V_2 and V_3, as well as between V_4 and V_5, but those are not part of the $MK_{3,3}$.

Corollary 3.9. *Every maximally planar graph with at least four vertices is 3-connected.*

Proof. Exercise. □

3.3 The proof of Kuratowski's theorem

Throughout this section, suppose that L is a non-planar graph that is minimal with respect to the minor relation, that is, every minor $G \neq L$ of L is planar. We shall show that $L = K^5$ or $L = K_{3,3}$, which will prove Wagner's (and thus also Kuratowski's) theorem.

Remark. *By Proposition 3.1 and the minimality of L, we know that L is 2-connected.*

If L has vertices of degree two, then L is a subdivision of a multigraph M with fewer vertices. Because L is non-planar, M is non-planar as well, and so is the graph obtained from M by deleting all multiple occurrences of edges. But this would contradict the minimality of L, thus $\delta(L) \geq 3$.

Since L is 2-connected, we know that it contains a cycle C. By the minimality of L, we know that all components of $L - C$ are planar. The basic idea is to start with a drawing of C and to attach all remaining components to C either in the inside face or in the outside face of this drawing. Clearly, the way that such components are attached to C can influence whether we are able to attach two components on the same "side" of C or not.

Definition. Let C be a cycle in a graph G. A *fragment* of G with respect to C is a graph $F = G[U] - E(C[U])$, where either

(i) $U = V(H) \cup N(V(H))$, where H is a component of $G - C$, or

(ii) $U = \{u, v\}$, where u, v are distinct vertices of C that are adjacent in G, but not in C.

The vertex set $A(F) := V(F \cap C)$ is the *attachment set* of F.

We say that two fragments F_1, F_2 *overlap* if

(i) there exist distinct vertices $x_1, y_1 \in A(F_1)$ and $x_2, y_2 \in A(F_2)$ that appear on C in the order x_1, x_2, y_1, y_2, or

(ii) $|A(F_1) \cap A(F_2)| \geq 3$.

The *graph of overlappings* $O_G(C)$ is the graph whose vertices are the fragments of G with respect to C, and in which two vertices are adjacent if and only if the fragments overlap.

Clearly, if two fragments overlap, then we cannot attach them to a given drawing of C on the same side of C. Thus, if $O_G(C)$ is not bipartite, then G cannot be planar. In fact, we will see that this is the only obstacle to planarity.

Lemma 3.10. *Suppose that C is a cycle in a graph G such that for each fragment F, the graph $C \cup F$ is planar. If $O_G(C)$ is bipartite, then G is planar.*

Proof. Fix a drawing of C. Let $F_1, \ldots, F_k$ be the fragments of G with respect to C and let (P, Q) be a bipartition of $\{F_1, \ldots, F_k\}$ such that no two fragments in the same partition class overlap. We shall prove by induction on i that $F_1 \cup \cdots \cup F_i$ can be added to the drawing of C for $0 \leq i \leq k$ such that all fragments in P are drawn into the inner face of C and all fragments in Q are drawn into the outer face. For $i = k$, this statement will then yield that G is planar.

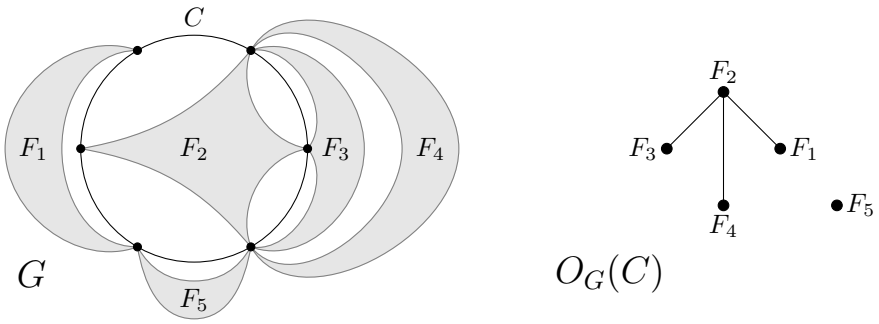

Figure 3.3: The fragments of a graph G with respect to a cycle C, and the graph of overlappings.

The induction basis is trivial. Now suppose that the induction hypothesis holds for some $0 \leq i \leq k-1$ and consider the fragment F_{i+1}. Without loss of generality, we may assume that $F_{i+1} \in P$. If there is an inner face of the drawing of

$$H := C \cup \bigcup_{j \leq i, F_j \in P} F_j$$

whose boundary contains $A(F_{i+1})$, then we can draw F_{i+1} into that face and be done.[6] So suppose that this is not the case. If there are two vertices $u, v \in A(F_{i+1})$ that do not lie on the boundary of a common inner face of H, then these vertices have to be separated by a fragment[7] in P, implying that this fragment overlaps with F_{i+1}. But this contradicts the fact that any two fragments in P do not overlap.

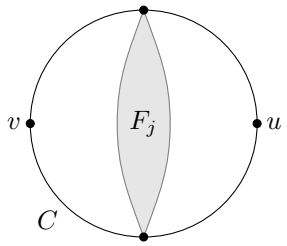

Figure 3.4: If two vertices u, v in the attachment set of F_{i+1} do not lie on the boundary of a common inner face of H, then they are separated by some earlier fragment $F_j \in P$, which then overlaps with F_{i+1}.

Now suppose that some three vertices $u, v, w \in A(F_{i+1})$ do not lie on the boundary of a common inner face of H. Then they have to be separated by some previous fragment $F_j \in P$, which means that either F_j separates two of the three vertices—which we already know to not be the case—or we have $u, v, w \in A(F_j)$,[8] again contradicting the fact that elements of P do not overlap.

[6] It is not completely trivial that drawing F_{i+1} into that face is automatically possible.
[7] This formally needs a statement similar to a reverse Jordan curve theorem.
[8] This part needs quite a couple of topological arguments to be made rigorous.

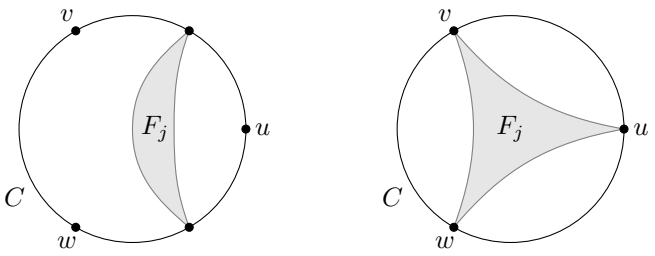

Figure 3.5: If three vertices u, v, w in the attachment set of F_{i+1} do not lie on the boundary of a common inner face of H, then there is some earlier fragment $F_j \in P$ that either separates two of the vertices (left) or contains all three in its attachment set. Either way, F_j overlaps with F_{i+1}.

If $A(F_{i+1})$ consists of at most three vertices, we are done, so assume that $A(F_{i+1}) = \{v_1, \dots, v_t\}$ with $t \geq 4$ and that $v_1, \dots, v_t$ do not lie on the boundary of a common inner face of H. Choose the indices so that $v_1, \dots, v_t$ appear on C in that order. Without loss of generality, there is an inner face f_1 of H whose boundary contains v_2, v_3, v_4 (and possibly some of the vertices $v_5, \dots, v_t$), but not v_1. Add to H a vertex u_1 drawn into f_1 and connect it to v_2, v_3, v_4, with the edges being drawn inside f_1 without crossings.

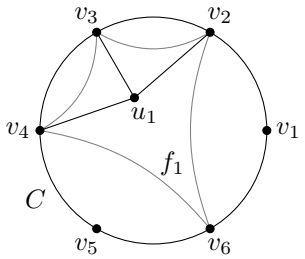

Figure 3.6: Adding the vertex u_1 to H.

We also know that there is an inner face f_2 of H whose boundary contains v_1, v_2, v_3. Add another vertex u_2 drawn into f_2 and connect it to v_1, v_2, v_3 inside f_2. Finally, add a vertex u_3 drawn into the outer face of H and connect it to v_1, v_2, v_3 inside the outer face. The nine edges we added do not cross and thus the resulting graph $\tilde{H}$ is plane. But $\tilde{H}$ contains a subdivision of $K_{3,3}$: both u_2 and u_3 are adjacent to each of v_1, v_2, v_3, while u_1 is adjacent to v_2, v_3 and is connected to v_1 by the path $u_1 v_4 C v_1$ that avoids the other five vertices. This contradicts the fact that $K_{3,3}$ is not planar. Thus, all vertices $v_1, \dots, v_t$ lie on the boundary of a common inner face of H, which proves the induction step. □

Lemma 3.11. *If C is a longest cycle in L, then for each fragment F, the graph $C \cup F$ is planar and the set $A(F)$ is independent in C. Furthermore, $O_L(C)$ is an odd cycle and every vertex in C is in the attachment set of some fragment.*

Proof. Let F be a fragment; we first show that $A(F)$ is independent in C. Suppose, for contradiction, that $u, v \in A(F)$ are adjacent on C. Then, because F is connected, there is a u–v path P in F. Moreover, P has length at least

two, because u, v are not adjacent in F by definition. But this means that we could make C longer by following P instead of using the edge uv, contradicting the maximality of C.

Secondly, it is clear, since $\delta(L) \geq 3$, that each vertex of C is in the attachment set of some fragment. In particular, since each $A(F)$ is independent in C, there must be at least two fragments. It follows that for any fragment F, $C \cup F$ is a proper subgraph of L and thus planar by the minimality of L.

In particular, Lemma 3.10 is applicable to C and hence the fact that L is not planar shows that $O_L(C)$ is not bipartite. On the other hand, if we choose any fragment F, then the union of C and all other fragments is planar by the minimality of L. Thus, deleting the vertex F from $O_L(C)$ results in a bipartite graph. The only non-bipartite graphs that become bipartite after the deletion of any single vertex are odd cycles.

□

For the remainder of this proof, suppose that we have fixed some longest cycle C in L. Denote the fragments of L with respect to C by $F_1, \ldots, F_k$. By Lemma 3.11, we may assume that $k \geq 3$ is odd and each F_i overlaps with F_{i-1} and F_{i+1} (where we write $F_0 = F_k$ and $F_{k+1} = F_1$) and with no other F_j's.

Lemma 3.12. *Each attachment set $A(F_i)$ has size two.*

Proof. By Lemma 3.11 no two consecutive vertices of C lie in the same attachment set and $\bigcup A(F_i) = V(C)$.

Since L is 2-connected, each attachment set has at least two vertices. Suppose, for contradiction, that some attachment set is larger, without loss of generality $|A(F_1)| \geq 3$. Let $x_1, \ldots, x_t$ be the vertices of $A(F_1)$, appearing in this order on C. For $i \neq j$, write $[x_i, x_j]$ for the vertex set of $x_i C x_j$.

For any $i = 1, \ldots, t$, if we delete from L all edges in F_1 incident with x_i, then the resulting graph is planar by the minimality of L and thus the graph of overlappings must have become bipartite. This means that F_2 or F_k does not overlap F_1 anymore, which implies $A(F_2) \subseteq [x_{i-1}, x_{i+1}]$ or $A(F_k) \subseteq [x_{i-1}, x_{i+1}]$ (where we again write $x_0 := x_t$ and $x_{t+1} := x_1$).

Suppose that $t = |A(F_1)| \geq 4$. Without loss of generality, we may assume that $A(F_2)$ is contained in $[x_t, x_2]$ and in $[x_1, x_3]$ or $[x_2, x_4]$. In the former case, we would have $A(F_2) \subseteq [x_t, x_2] \cap [x_1, x_3] = [x_1, x_2]$, which contradicts the fact that F_1 and F_2 overlap in L. In the latter case, we have

$$A(F_2) \subseteq [x_t, x_2] \cap [x_2, x_4] = \begin{cases} \{x_2, x_4\} & \text{if } t = 4, \\ \{x_2\} & \text{otherwise.} \end{cases}$$

Since $A(F_2)$ has at least two vertices, this would imply $t = 4$ and $A(F_2) = \{x_2, x_4\}$. By the same arguments, we deduce that $A(F_k) = \{x_1, x_3\}$. This way, F_2 and F_k overlap, which means that $k = 3$. But then

$$V(C) = A(F_1) \cup A(F_2) \cup A(F_3) = A(F_1),$$

contradicting the fact that no consecutive vertices of C lie in the same attachment set by Lemma 3.11.

We may thus assume that $t = 3$ and that all other attachment sets have size *at most* three. We have three intervals $[x_1, x_3]$, $[x_2, x_1]$, and $[x_3, x_2]$, each of

which contains, again by the minimality of L, (at least) one of the sets $A(F_2)$ and $A(F_k)$. Without loss of generality, $A(F_2)$ is contained in $[x_1, x_3]$ and $[x_2, x_1]$ and thus in

$$[x_1, x_3] \cap [x_2, x_1] = \{x_1\} \cup [x_2, x_3].$$

In particular, $x_1 \in A(F_2)$, because otherwise F_1 and F_2 would not overlap.

If x_2 and x_3 both lie in $A(F_2)$ as well, then $A(F_2) = A(F_1)$ (because $A(F_2)$ cannot be larger than $A(F_1)$). As F_k overlaps F_1, it overlaps F_2 as well, which again means that $k = 3$. Since $A(F_1)$ is an independent set in C, each of the three segments $[x_i, x_{i+1}]$ has at least one internal vertex y_i, which has to lie in some attachment set. Since they can neither lie in F_1 nor in F_2, we have $A(F_3) = \{y_1, y_2, y_3\}$. But then neither $A(F_1)$ nor $A(F_2)$ is contained in $[y_1, y_3]$, which would have to be the case if we exchange the roles of F_1 and F_3.

Therefore, not both x_2 and x_3 lie in $A(F_2)$, say $x_3 \notin A(F_2)$. In order for F_1 and F_2 to overlap, $A(F_2)$ has to contain some internal vertex y_2 from $[x_2, x_3]$. In particular, $A(F_2) \not\subseteq [x_3, x_2]$ und thus $A(F_k) \subseteq [x_3, x_2]$. As F_1 and F_k overlap, F_2 and F_k overlap as well and we have $k = 3$. All internal vertices of $[x_1, x_2]$ and of $[x_3, x_1]$ have to lie in $A(F_3)$. Hence, each of the two intervals has only one internal vertex, because $A(F_3)$ contains no consecutive vertices on C. Similarly, $A(F_3) \subseteq [x_3, x_2]$ implies that all internal vertices of $[x_2, x_3]$ lie in $A(F_2)$, which means that y_2 is the only such vertex. Thus, we have $V(C) = \{x_1, y_1, x_2, y_2, x_3, y_3\}$ (in this order), $A(F_2) = \{x_1, y_2\}$, and $A(F_3) = \{y_1, y_3\}$. But then setting

$$\begin{aligned} S_1 &:= \{x_1\}, & S_2 &:= \{x_2\}, & S_3 &:= \{x_3\}, \\ S_4 &:= V(F_1) \setminus A(F_1), & S_5 &:= V(F_2) \setminus \{x_1\}, & S_6 &:= V(F_3), \end{aligned}$$

results in connected sets $S_1, \ldots, S_6$ with $L/(S_1, \ldots, S_6) = K_{3,3}$ (with sides $\{S_1, S_2, S_3\}$ and $\{S_4, S_5, S_6\}$), contradicting the minimality of L, because at least the set S_6 contains more than one vertex. This proves that all attachment sets have size two. □

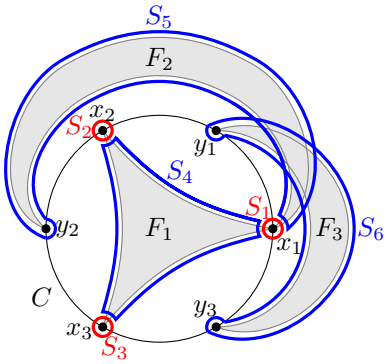

Figure 3.7: Finding an $MK_{3,3}$ in L if F_1 has attachment set $\{x_1, x_2, x_3\}$.

Lemma 3.13. *Each F_i is an edge.*

Proof. Since each attachment set has size 2, we can contract each fragment F to a single edge and obtain a graph L' with $O_{L'}(C) = O_L(C)$. In particular, since L is non-planar, $O_L(C)$ is not bipartite and so by Lemma 3.10 neither is L', contradicting the minimality of L unless each fragment is an edge. □

If $k = 3$, we know that F_1, F_2, F_3 pairwise overlap, which by Lemmas 3.11 and 3.13 implies that C is a cycle of length six, with vertices $v_1, \dots, v_6$ say (in this order), and each F_i is an edge between v_i and v_{i+3}. But then $L = K_{3,3}$ (with sides $\{v_1, v_3, v_5\}$ and $\{v_2, v_4, v_6\}$) and we would be done. We may thus assume from now on that $k \geq 5$.

Lemma 3.14. *For each F_i, one of the two cycles of $C \cup F_i$ that go through F_i contains all attachment sets $A(F_j)$ with $j \notin \{i-1, i, i+1\}$; the other cycle contains no attachment set apart from $A(F_i)$.*

Proof. Observe first that the fragments F_{i-1} and F_{i+1} overlap with F_i and thus their attachment sets are contained in neither of the two cycles. Each other fragment does not overlap with F_i and thus its attachment set has to be contained in one of the two cycles.

Suppose now that a cycle $C' \neq C$ in $C \cup F_i$ contains $A(F_j)$ for some $j \neq i$. Then F_{j+1} overlaps with F_j and thus $A(F_{j+1})$ has one vertex in C'. Unless $j + 1 = i - 1$, F_{j+1} and F_i will not overlap and thus $A(F_{j+1})$ is contained in C'. Inductively, we get that $A(F_{j+1}), A(F_{j+2}), \dots, A(F_{i-2})$ lie in C' and by symmetry, the same holds for $A(F_{j-1}), A(F_{j-2}), \dots, A(F_{i+2})$. This proves the lemma. □

For each i, denote by C_i the cycle from Lemma 3.14 that contains no attachment set. Each C_i contains one vertex from $A(F_{i-1})$ and $A(F_{i+1})$ each, thus C_i has length three (if these two vertices coincide) or four. We can denote the end vertices of each F_i by x_i and y_i so that each C_i consists of x_i, y_{i-1}, x_{i+1}, and y_i (in that order, but possibly with $x_{i+1} = y_{i-1}$).

Set

$$
\begin{aligned}
S_1 &:= \{x_3, y_1\},\\
S_2 &:= \{x_4, y_2\},\\
S_3 &:= \{x_5, y_3\},\\
S_4 &:= \{x_1\} \cup \left\{x_{2i} \colon 3 \leq i \leq \frac{k-1}{2}\right\} \cup \left\{y_{2j} \colon 2 \leq j \leq \frac{k-1}{2}\right\},\\
S_5 &:= \{x_2\} \cup \left\{x_{2i+1} \colon 3 \leq i \leq \frac{k-1}{2}\right\} \cup \left\{y_{2j+1} \colon 2 \leq j \leq \frac{k-1}{2}\right\}.
\end{aligned}
$$

(Note that S_1 will consist of just one vertex if $x_3 = y_1$, and the same holds for S_2 and S_3.) Then $S_1, \dots, S_5$ are connected and satisfy $L/(S_1, \dots, S_5) = K^5$, a contradiction to the minimality of L, unless $k = 5$ and $x_{i+1} = y_{i-1}$ for all i, in which case we have $L = K^5$.

Summing up, we derive a contradiction to the minimality of L in all cases apart from $L = K_{3,3}$ or $L = K^5$. This proves that $K_{3,3}$ and K^5 are the only minor-minimal non-planar graphs. As every non-planar graphs contains a minor-minimal one as a minor, we have thus proved Wagner's theorem—and thus by Proposition 3.8 also Kuratowski's theorem.

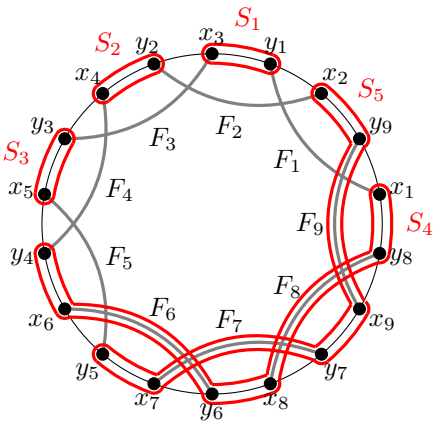

Figure 3.8: Finding an MK^5 in L. For each i, the vertices y_{i-1} and x_{i+1} may coincide.

3.4 Graphs on other surfaces

Instead of drawing a graph in the plane, one could equivalently draw it on a sphere. An obvious generalisation of this concept is drawings of graphs on any closed surface, e.g. the torus, the projective plane, the Klein bottle etc. A graph that can be drawn on some given surface is called *embeddable* on this surface. Given some drawing, faces and their boundaries are defined analogously to the planar case.

Regarding drawings of graphs, it is often most convenient to represent a surface by its *fundamental polygon*, a polygon of even length whose edges are (pointwise) identified in pairs to give rise to the desired surface.

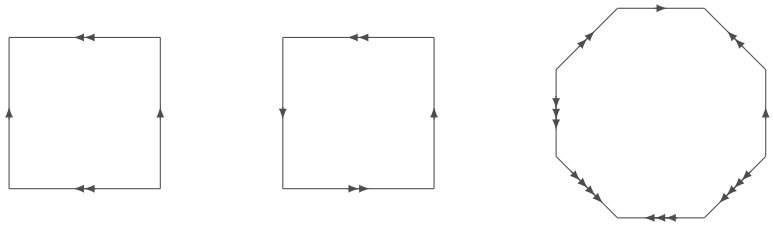

Figure 3.9: Fundamental polygons of the torus (left), the projective plane (middle), and the double torus (right). Edges with the same arrow type are identified with the orientation indicated by the arrows.

For instance, if we consider the torus, then all planar graphs are clearly still embeddable. The faces, however, do not have to be of the form that they are on the sphere. More specifically, the faces of a connected plane graph are always homeomorphic to open discs (that is, homeomorphic to the open set $\{x \in \mathbb{R}^2 \colon \|x\| < 1\}$ in $\mathbb{R}^2$), but if we draw, say, K^4 on the torus, we might end up with several types of faces.

What about drawing non-planar graphs on other surfaces? For the torus, K^5 and $K_{3,3}$ become embeddable, and even K^6 and K^7 can now be drawn.

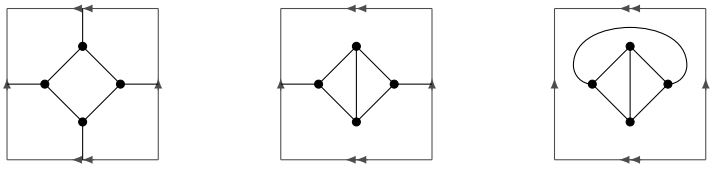

Figure 3.10: Three different ways to draw K^4 on a torus. In the situation on the left, we have two faces, both of which are discs. In the middle, two of the three faces are discs and the third one is an open cylinder. On the right, three faces are discs and the last face is a torus with a hole.

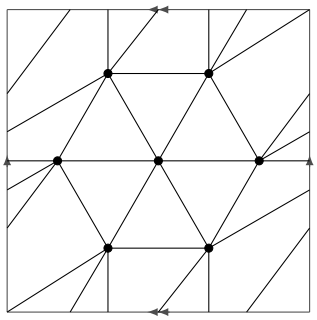

Figure 3.11: Drawing K^7 on the torus.

This clearly shows that Euler's formula is not true for the torus (in fact, it fails for all surfaces apart from the sphere) and so is the upper bound on the number of edges from Proposition 3.3. However, we have analogous results to the planar case for arbitrary (closed) surfaces. We shall state all results in this section without proof.

Proposition 3.15. *Let S be a closed surface. There exists a constant $g \geq 0$ such that*

(i) every graph G drawn on S has at least

$$\|G\| - |G| + 2(1 - g)$$

faces, with equality if and only if all faces are homeomorphic to open discs;[9]

(ii) each graph with $n \geq 3$ vertices that is embeddable on S has at most

$$3n + 6(g - 1)$$

edges.

Just like for planar graphs, the upper bound in (ii) follows directly from (i).

The value g from Proposition 3.15 is also called the *genus* of the surface.[10] The sphere has genus 0, the torus has genus 1, the double torus genus 2, and

[9] Observe that this can only be the case when G is connected.

[10] The notion of the genus can be different in different contexts. A topology textbook for instance will most probably feature a different definition than ours.

so on. Among the non-orientable surfaces, the projective plane has the smallest genus, namely $\frac{1}{2}$. The Klein bottle has genus 1.

As for planar graphs, embeddability on a surface is closed under taking minors or topological minors. It is thus natural to ask what properties the *minimal* (with respect to the minor relation or the topological minor relation) non-embeddable graphs have.

Definition (Forbidden minor)**.** Let S be a closed surface. A graph G is called *forbidden minor for* S if G is not embeddable on S, but every proper minor of G is embeddable. *Forbidden topological minors* are defined analogously.

For the sphere, Wagner's theorem and Kuratowski's theorem tell us that the only forbidden minors or forbidden topological minors are K^5 and $K_{3,3}$. What about other surfaces? Do we know which graphs are forbidden minors, or do we at least know the number of forbidden minors? A priori, it is not even clear that there should only be a finite number of forbidden minors.

Theorem 3.16 (Robertson, Seymour)**.** *In any infinite set of graphs, there are two graphs such that one of them is a minor of the other.*

The proof of Theorem 3.16 is spread over 20 papers called "Graph Minors. I" (published 1983) to "Graph Minors. XX" (from 2004) and consists of more than 500 pages in total.[11] The work of Robertson and Seymour has provided a plethora of new concepts, results, and insights.

As a corollary, we get immediately that for any surface, the set of forbidden minors is finite.

Corollary 3.17. *For every closed surface S, the set of forbidden minors is finite.*

For topological minors, an analogue of Theorem 3.16 does *not* hold—there are infinite sets of graphs in which none is a topological minor of another. Corollary 3.17, however, remains true if we replace "forbidden minors" by "forbidden topological minors" (exercise).

For the projective plane—the surface with the smallest non-zero genus—the sets of forbidden minors and forbidden topological minors have been determined; there are 35 forbidden minors and 103 forbidden topological minors. But even for the torus, seemingly the "easiest" remaining case, the full lists are not known. So far, 16629 forbidden minors and 239322 forbidden topological minors have been found for the torus. This makes the theorems of Kuratowski and Wagner all the more impressive.

3.5 Planarity recognition algorithms

A planarity recognition algorithm should tell us, for any input graph G, whether G is planar, preferably within short running time. An additional feature would be if the algorithm also outputs a reason for its decision, i.e. a drawing of G if G is planar, or some certificate for the non-planarity of G otherwise.

[11] Since then, Robertson and Seymour published more papers under the "Graph Minors" header, the most recent being "Graph Minors. XXIII" from 2010. These newer works are part of the "Graph minors project", but the proof of Theorem 3.16, by far the most famous result of the project, was finished with Part XX.

Using the theorems of Kuratowski and Wagner, one could try to look for K^5 and $K_{3,3}$ as (topological) minors. All algorithms relying on this approach have the downside that they do *not* provide a drawing if the graph is planar.

Naïve approach. If we contract edges in all possible orders and check whether we end up with a K^5 or a (graph containing a) $K_{3,3}$, we need time $O(m!)$, which obviously is rather bad.

Naïve approach 2. Decompose the vertex set into five or six sets and check whether they are connected. If so, contract them and check whether the obtained minor is a K^5 or a (graph containing a) $K_{3,3}$. This algorithm can be implemented to have running time $O(n^2 6^n)$, which is much better than a running time of order $m!$, but still exponential.

More sophisticated approach. Results of Robertson and Seymour from the "Graph Minors" papers imply that for any fixed graph H, it only takes time $O(n^2)$ to check whether G contains an MH. In particular, planarity can be tested in quadratic time. Unfortunately, the description of the algorithm needs a couple of concepts and results about graph minors, which is why we will not present any details about that algorithm in this course.

Finding a drawing. Our proof of Kuratowski's theorem can be used to create an algorithm that checks planarity and either finds a drawing or outputs a certificate for the graph not being planar. We start by finding a cycle C.

- If $F \cup C$ is planar for each fragment F (which is checked by recursive application of the algorithm), and if the graph $O_G(C)$ of overlappings is bipartite, then we can recursively combine the drawings of the fragments to a drawing of G.
- Otherwise, the algorithm will find a certificate for the non-planarity of G, namely a non-bipartite graph of overlappings (not necessarily $O_G(C)$, but $O_{G'}(C')$ for some subgraph G' of G appearing in the recursive applications of the algorithm).

This algorithm is reasonably fast (exercise), and even faster if we assume that there is a cycle C with $V(C) = V(G)$. Furthermore, if we're a little more careful and find a cycle C where the attachment set of each fragment F is independent in C (for example a longest cycle), then the argument in the proof will even find an MK^5 or $MK_{3,3}$.

Linear time algorithms. There are different algorithms that can check planarity (and find a certificate for or against it) in time $O(n)$. The first such algorithm was presented by Hopcroft and Tarjan in 1974. However, linear time planarity tests are not exactly easy to understand.

> The known planarity algorithms that achieve linear time complexity are all difficult to understand and implement. This is a serious limitation for their use in practical systems. A simple and efficient algorithm for testing the planarity of a graph and constructing planar representations would be a significant contribution.

[G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: an annotated bibliography*, Computational Geometry 4 (1994), 235–282]

More on the algorithmic aspects of planar graphs can for instance be found in Chapter 8 of *The LEDA Platform of Combinatorial and Geometric Computing* by K. Mehlhorn and S. Näher, which is available online at

`https://people.mpi-inf.mpg.de/~mehlhorn/LEDAbook.html`

Chapter 4

Colourings

Definition (Colouring, chromatic number)**.** A *colouring* of a graph G is a function $c\colon V(G) \to S$, for which $uv \in E(G)$ implies $c(u) \neq c(v)$ (in words: adjacent vertices get different colours). S is called the set of *colours*. For each $s \in S$, we call the set $c^{-1}(s)$ a *colour class*.

A colouring with k colours is called *k-colouring*. If a k-colouring of G exists, we also say that G is *k-colourable*. If c is a k-colouring, we will often tacitly assume that the set of colours is $\{1, \ldots, k\}$.

We define the *chromatic number* of G as $\chi(G) := \min\{k\colon G \text{ is } k\text{-colourable}\}$.

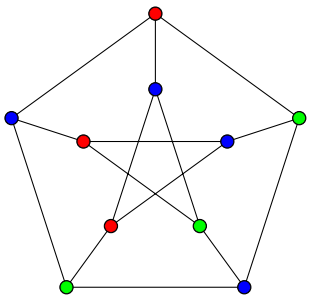

Figure 4.1: A 3-colouring of the Petersen graph.

4.1 Basic bounds

Can we give upper or lower bounds on the chromatic number? The number of vertices is clearly an upper bound, and 2 is a lower bound if the graph has at least one edge. Less trivial bounds can be obtained by comparing the chromatic number with other graph parameters.

Definition (Clique number, independence number)**.** The *clique number* $\omega(G)$ of a graph G is the size of a largest clique in G. The *independence number* $\alpha(G)$ of G is the size of the largest independent set.

The clique number and the independence number provide easy lower bounds on the chromatic number.

Proposition 4.1. *For every non-empty graph G, the following statements hold.*

(i) $\chi(G) \geq \omega(G)$;

(ii) In any colouring of G, each colour class is an independent set;

(iii) $\chi(G) \geq \left\lceil \frac{|G|}{\alpha(G)} \right\rceil$.

Proof. (i) G contains a $K^{\omega(G)}$ and thus $\chi(G) \geq \chi(K^{\omega(G)}) = \omega(G)$.

(ii) Any edge between two vertices from the same colour class would contradict the definition of a colouring.

(iii) If there were a k-colouring of G with $k < \frac{|G|}{\alpha(G)}$, then by the pigeonhole principle one of the colour classes would contain at least $\frac{|G|}{k} > \alpha(G)$ vertices. However, by (ii) each colour class is independent, a contradiction. □

Observe that the lower bounds (i) and (iii) can be tight in some cases and far off the truth in other cases. There exist

- graphs with $\chi(G) = \omega(G) = |G| / \alpha(G)$ (e.g. even cycles);
- graphs with $\chi(G) = \omega(G)$, but $|G| / \alpha(G)$ being much smaller (e.g. a complete graph plus "many" isolated vertices);
- graphs with $|G| / \alpha(G)$ "close" to $\chi(G)$, but $\omega(G)$ being much smaller (see Proposition 4.2);
- graphs with both $\omega(G)$ and $|G| / \alpha(G)$ much smaller than $\chi(G)$.

Proposition 4.2. *For every positive integer k, there exists a triangle-free graph G_k with $\chi(G_k) = k$.*

Proof. For $k = 1, 2$, the graphs $G_1 := K^1$ and $G_2 := K^2$ suffice. Now suppose that $k \geq 3$ and that we already know G_{k-1} with the desired properties. Write $V(G_{k-1}) = \{v_i \colon 1 \leq i \leq |G_{k-1}|\}$. For each v_i, add to G_{k-1} a vertex w_i and connect it to all neighbours of v_i in G_{k-1}. Finally, add a vertex z and connect it to all w_i's. Denote the resulting graph by G_k.

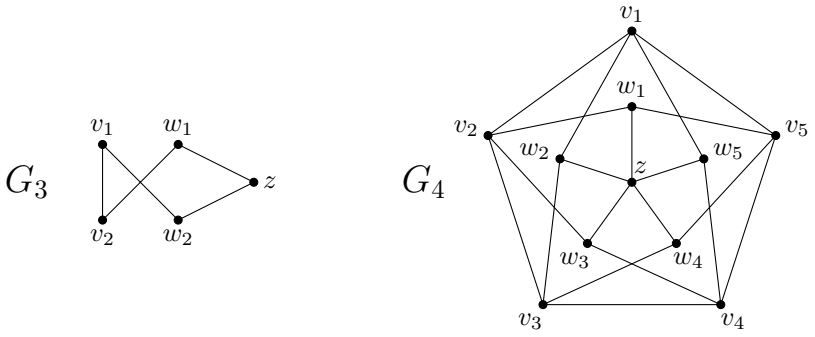

Figure 4.2: Constructing G_k from G_{k-1}.

Let us first show that G_k is triangle-free. We know that G_{k-1} is triangle-free and that the w_i's form an independent set. Since z is only adjacent to w_i's, this means that any triangle in G_k would have to be of the form $w_i v_j v_\ell$. But then $v_i v_j v_\ell$ would be a triangle in G_{k-1}, a contradiction.

By assumption there is a $(k-1)$-colouring c of G_{k-1}. We can define a k-colouring of G_k by giving each v_i and w_i the colour $c(v_i)$ and giving z the colour k. It is easy to check this is a colouring of G, and hence, $\chi(G_k) \leq k$.

It remains to prove that G_k is *not* $(k-1)$-colourable. Suppose, for contradiction, that there exists a $(k-1)$-colouring c of G_k and set $j := c(z)$. Consider the colouring that c induces on G_{k-1}. If a vertex of colour j does not have neighbours (in G_{k-1}) of *all* other colours, then we could re-colour it with some colour missing among its neighbours and still have a colouring of G_{k-1}. If this were possible for all vertices coloured with j, we could completely avoid using colour j. This would give us a $(k-2)$-colouring of G_{k-1}, which we know not to exist.

This means that there is some vertex v_{i_j} of colour j with all other colours appearing among the neighbours of v_{i_j} in G_{k-1}. Since w_{i_j} is adjacent to all these vertices, we have $c(w_{i_j}) = j$ as well. But then w_{i_j} is a neighbour of z with the same colour as z, a contradiction. Thus, $\chi(G_k) = k$. □

The graph G_k from Proposition 4.2 has clique number $\omega(G_k) = 2$. By construction, we have $|G_k| = 2|G_{k-1}| + 1$ and $\alpha(G_k) \geq |G_{k-1}|$ for $k \geq 3$, which means that $\lceil |G_k| / \alpha(G_k) \rceil \leq 3$. (With a little effort, one can even show that $\alpha(G_k) = |G_{k-1}|$ and thus $\lceil |G_k| / \alpha(G_k) \rceil = 3$.) In particular, both lower bounds from Proposition 4.1 are far from the true value $\chi(G_k) = k$.

Remark. *An even stronger statement than Proposition 4.2 is that for every $k \geq 3$, there exists a graph H_k with $\chi(H_k) \geq k$, but with no cycle of length k or shorter.*

These graphs are much harder to construct directly, and their existence is usually proved using random graphs. The course Probabilistic method in combinatorics and algorithmics *features a proof.*

Let us now determine an upper bound on $\chi(G)$. The easiest bound is simply derived by counting edges.

Proposition 4.3. $\chi(G) \leq \frac{1}{2} + \sqrt{2\|G\| + \frac{1}{4}}$ *for every graph G.*

Proof. In a $\chi(G)$-colouring c, there has to be at least one edge between any two colour classes $c^{-1}(i), c^{-1}(j)$, because otherwise we could simply recolour all vertices in $c^{-1}(j)$ with colour i to find a colouring with less colours, a contradiction to the definition of $\chi(G)$. Thus, $\|G\| \geq \binom{\chi(G)}{2} = \frac{1}{2}(\chi(G)^2 - \chi(G))$. Solving this inequality for $\chi(G)$ yields the claimed bound. □

Similar to planarity, the chromatic number can be determined by looking at all blocks of a graph.

Proposition 4.4. $\chi(G) = \max\{\chi(B) \colon B \text{ is a block of } G\}$ *for every graph G.*

Proof. "$\geq$" is clear. Vice versa, observe that by Proposition 1.13, we can denote the blocks of G by $B_1, \ldots, B_k$ so that each B_i meets $B_1 \cup \cdots \cup B_{i-1}$ in at most one vertex, which we then call v_i. Let us write $m := \max\{\chi(G_1), \ldots, \chi(G_k)\}$.

Recursively colour each B_i with m colours. If necessary, permute the colours in such a way so as to assign to v_i the colour it has in $B_1 \cup \cdots \cup B_{i-1}$. Inductively, this defines an m-colouring of G. □

4.2 Colouring planar graphs

The most famous result about graph colourings is arguably the "Four Colour Theorem".

Theorem 4.5 (Appel, Haken 1977)**.** *Every planar graph is 4-colourable.*

The Four Colour Theorem is mostly known for its computer-based proof. Later, a written proof comprising 741 pages was published. The general proof strategy can be summarised as follows.

(i) Assume that we are given a drawing of G, which we may assume is maximally planar, and so this drawing is a triangulation.

(ii) Prove that G then contains (at least) one of 1476 so-called "unavoidable configurations".

(iii) Show that any triangulation containing such a configuration is "reducible", that is, it can be decomposed into smaller parts so that any 4-colourings of these parts can be combined into a 4-colouring of G. Each configuration has to be treated separately, leading to a case distinction with 1476 cases, which was solved using a computer.[1]

(iv) Use induction on $|G|$.

Before the Four Colour Theorem was proved, it was conjectured to be true for more than hundred years and many (false) proofs were published. One of these attempts led to a proof of the "Five Colour Theorem".

Theorem 4.6. *Every planar graph is 5-colourable.*

Proof. Suppose, for contradiction, that there are planar graphs that are not 5-colourable; let G be a smallest such graph. Clearly, G has at least six vertices and thus at most $3\,|G|-6$ edges by Proposition 3.3. This means that the average degree of G is smaller than six and so there is a vertex v with $d(v) \leq 5$. By the minimality of G, the graph $G - v$ is 5-colourable.

Fix a 5-colouring c of $G - v$. If the neighbours of v are coloured with four (or less) colours, then we could use the fifth colour for v, contradicting the fact that G is not 5-colourable. In particular, v has precisely five neighbours and each is coloured with a different colour. Fix some drawing of G and denote the neighbours of v by $x_1, \ldots, x_5$ so that they appear around v in that order (in the drawing). Without loss of generality, we may assume that $c(x_i) = i$ for $i = 1, \ldots, 5$.

For $1 \leq i \leq 5$, let us write $U_i := c^{-1}(i)$. Let H_1 be the component of the graph $G[U_1 \cup U_3]$ (i.e. the graph induced on the vertices of colours 1 and 3) that contains x_1. If x_3 does not lie in H_1, then we can exchange the colours 1

[1] In 1996, Robertson, Sanders, Seymour, and Thomas gave an alternative proof which only needed 633 configurations, but still relied on computer assistance to prove their reducibility.

and 3 for all vertices in H_1 and create a new colouring that does not use the colour 1 for the neighbours of v. In that case, we give colour 1 to v and have a 5-colouring of G, a contradiction.

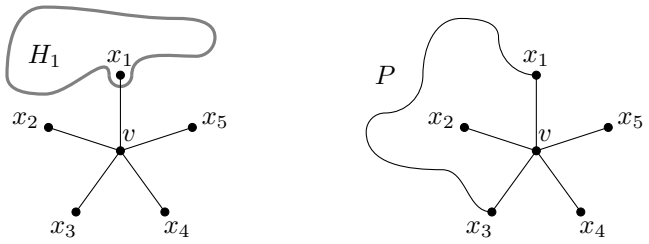

Figure 4.3: The first proof of the Five Colour Theorem. Either we can switch colours 1 and 3 in H_1, resulting in colour 1 not being used among the neighbours of v (left), or there exists an x_1–x_3 path P in colours 1 and 3, which (together with the edges vx_1 and vx_3) separates x_2 and x_4.

Thus, $G[U_1 \cup U_3]$ contains an x_1–x_3 path P. Analogously, we find an x_2–x_4 path Q in $G[U_2 \cup U_4]$. But this is not possible, since the cycle $x_1Px_3vx_1$ in G is disjoint from Q, but separates x_2 and x_4. □

Alternative proof. Like in the first proof, suppose that G is a minimal counterexample. Again, we find a vertex v of degree five. As G is planar, it does not contain a K^5. In particular, the neighbours of v are not pairwise adjacent. Suppose that neighbours w_1, w_2 of v are not adjacent. Then $G' := G/\{v, w_1, w_2\}$ is a minor of G and hence planar. Thus, G' is 5-colourable due to the minimality of G.

Pick a 5-colouring of G'. This induces a 5-colouring of $G - v$ by giving w_1 and w_2 the colour of $\{v, w_1, w_2\}$ and letting all other vertices keep their colour. As w_1, w_2 have the same colour, at most four colours are used for the neighbours of v and we can colour it with the fifth colour to obtain a 5-colouring of G, a contradiction to the choice of G. □

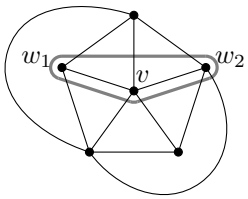

Figure 4.4: The second proof of the Five Colour Theorem. The graph G' obtained by contracting $\{v, w_1, w_2\}$ is planar and thus 5-colourable by the minimality of G. A 5-colouring c' of G' induces a 5-colouring c of $G - v$ by setting $c(w_1) = c(w_2) := c'(\{v, w_1, w_2\})$ and $c(x) := c'(x)$ for all other vertices. Then at least one colour is "free" to be used for v.

Finally, let us state (without proof) another well-known result.

Theorem 4.7 (Grötzsch 1959)**.** *Every triangle-free planar graph is 3-colourable.*

4.3 The greedy algorithm

If we want an algorithm that finds a colouring for every input graph G, there are two opposing aims. The first aim would be that the colouring uses "few" colours (preferably close to $\chi(G)$); the second aim would be a "short" running time. Let us first concentrate on running time.

Suppose that the vertex set of our input graph is given with some fixed ordering $v_1, \dots, v_n$ of the vertices. The basic idea of our first algorithm is to take, in each step, the first vertex that has no colour yet, and assign to it the first colour that is not already used for any of its neighbours.

```
PROCEDURE GREEDY_COLOURING
BEGIN
   c(v_1) := 1;
   FOR (i = 2 to n){
      c(v_i) := min{k ∈ ℕ \ {0}: k ≠ c(v_j) ∀j < i with v_j ∈ N(v_i)};
   }
END
```

Definition (Greedy algorithm)**.** We call the algorithm above the *greedy algorithm*. For an ordering $\sigma = (v_1, \dots, v_n)$ of the vertices of a graph G, we denote by $\chi_{\mathrm{Gr}}(G, \sigma)$ the number of colours used by the greedy algorithm.

Clearly, the greedy algorithm produces a valid colouring. This colouring, however, might use more than $\chi(G)$ colours. For instance, depending on the ordering of the vertices, the greedy algorithm might need three colours to colour a cycle of length six.

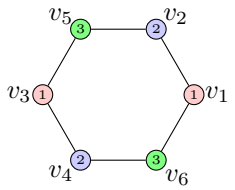

Figure 4.5: Colouring a cycle of length six with the greedy algorithm. If the vertices are numbered as in the picture, then the greedy algorithm will need three colours.

Proposition 4.8. *Let σ be a fixed ordering of $V(G)$, where $G \neq \emptyset$.*

(i) The greedy algorithm runs in time $\Theta(n + m)$.

(ii) $\chi(G) \leq \chi_{\mathrm{Gr}}(G, \sigma) \leq 1 + \Delta(G)$.

Proof. (i) Every vertex is considered as v_i exactly once, and every edge $e = v_i v_j$ with $j < i$ adds exactly one calculation step (when the minimum in the definition of $c(v_i)$ is determined).

(ii) The lower bound is trivial, since the algorithm produces a valid colouring. For the upper bound, observe that each v_i has at most $d(v_i) \leq \Delta(G)$ neighbours amongst $v_1, \dots, v_{i-1}$ and thus $c(v_i) \leq 1 + \Delta(G)$. □

There are graphs for which the lower bound $\chi(G)$ and the upper bound $1 + \Delta(G)$ in (ii) coincide (e.g. complete graphs or odd cycles). On the other hand, the two values can be arbitrarily far apart, and $\chi_{\mathrm{Gr}}(G, \sigma)$ can be at either end of the spectrum.

It is easy to see that there is always an ordering resulting in the greedy algorithm needing only $\chi(G)$ colours (exercise). However, in terms of an efficient algorithm, we would need to be able to identify a "good" ordering in short time, without having any additional information. To this end, observe that when the greedy algorithm assigns a colour to v_i, it seems to be an advantage if v_i has as few neighbours amongst $v_1, \ldots, v_{i-1}$ as possible.

Definition (Backwards degree)**.** Given an ordering σ of $V(G)$, let us write $G_i := G[v_1, \ldots, v_i]$. The *backwards degree* of v_i is defined as $d_{G_i}(v_i)$. We denote by $b(\sigma) := \max_{1 \leq i \leq n} d_{G_i}(v_i)$ the largest backwards degree with respect to σ.

Observe that the upper bound in Proposition 4.8(ii) can be improved using the notation of the backwards degree, because for each v_i, only its degree in G_i is important. We thus immediately have the following result.

Proposition 4.9. *For every graph G, we have $\chi(G) \leq 1 + \min_\sigma b(\sigma)$, where the minimum is taken over all orderings of $V(G)$.* □

Can we find an ordering σ that minimises $b(\sigma)$ without having to check all possible orderings? Surprisingly, this is even possible in linear time.

```
PROCEDURE SMALLEST_LAST
BEGIN
   G_n := G;
   FOR (i = n down to 1){
      Find vertex v_i of minimum degree in G_i;
      G_{i-1} := G_i − v_i;
   }
END
```

Remark. *Given G, we can determine all vertex degrees in time $\Theta(n + m)$. Encode the degrees by saving, for each i, all vertices of degree i in a list. Then indentifying v_i takes time $\Theta(d_{G_i}(v_i))$ and so does adjusting the degrees for G_{i-1}. This means that it takes time $\Theta(\sum_i d_{G_i}(v_i)) = \Theta(n + m)$ to finish all instances of the FOR-loop.*

Thus, SMALLEST_LAST can be implemented to run in time $\Theta(n + m)$.

We use the natural notion of a *smallest-last ordering* for any ordering that can be produced by the smallest-last algorithm.

Proposition 4.10. *For any graph G and any smallest-last ordering σ_0 of $V(G)$, we have $b(\sigma_0) = \max_{H \subseteq G} \delta(H) = \min_\sigma b(\sigma)$.*

Proof. By construction, we have $b(\sigma_0) = \max_i \delta(G_i) \leq \max_{H \subseteq G} \delta(H)$. Suppose that $H = H_0$ maximises $\delta(H)$. For any ordering σ, let k be the largest index of a vertex in H_0. Then

$$\max_{H \subseteq G} \delta(H) = \delta(H_0) \leq d_{H_0}(v_k) \leq d_{G_k}(v_k) \leq b(\sigma)$$

and thus

$$b(\sigma_0) \leq \max_{H \subseteq G} \delta(H) \leq \min_{\sigma} b(\sigma) \leq b(\sigma_0),$$

which means that we have equality, as desired. □

With this result, we can improve the upper bound from Proposition 4.8(ii) in the case of a smallest-last ordering σ.

Corollary 4.11. *Let G be any graph.*

(i) $\chi(G) \leq \chi_{\mathrm{Gr}}(G, \sigma) \leq 1 + \max_{H \subseteq G} \delta(H)$ for every smallest-last ordering σ of $V(G)$.

(ii) G contains a subgraph of minimum degree at least $\chi(G) - 1$.

Proof. (i) follows directly from Propositions 4.9 and 4.10. (ii) is an immediate consequence of (i). □

4.4 Brooks' theorem

In some cases (e.g. $K_{1,r}$ as an extreme example), the upper bound for the chromatic number from Corollary 4.11 is much better that the immediate bound $1 + \Delta(G)$ from Proposition 4.8(ii). In some other cases (e.g. regular graphs), however, the two bounds coincide. The next result shows that being regular is essentially the only cause for the bounds to coincide.

Proposition 4.12. *If G is connected and non-regular, then $\chi(G) \leq \Delta(G)$.*

Proof. Since G is not regular, we have $\delta(G) \leq \Delta(G) - 1$. Let v be a vertex of degree $\delta(G)$ and let H be any subgraph of G. If v lies in H, then $\delta(H) \leq d_H(v)$ and thus in particular $\delta(H) \leq \Delta(G) - 1$. Otherwise, due to the connectedness of G, there is a vertex w in H with a neighbour in $G - V(H)$. Thus,

$$\delta(H) \leq d_H(w) \leq d_G(w) - 1 \leq \Delta(G) - 1.$$

Therefore, $\chi(G) \leq \Delta(G)$ by Corollary 4.11. □

As mentioned before, complete graphs and odd cycles are examples for graphs with $\chi(G) = \Delta(G) + 1$. These are basically the only such graphs.

Theorem 4.13 (Brooks 1941)**.** *If G is connected and neither complete nor an odd cycle, then $\chi(G) \leq \Delta(G)$.*

There are direct proofs of Brooks' theorem (see e.g. Diestel's book), but we will prove it via an auxiliary result.

Lemma 4.14. *If G is 2-connected and neither complete nor a cycle, then there are two vertices x, y of G of distance two (i.e. x, y are not adjacent, but have a common neighbour), for which $G - x - y$ is connected.*

Proof. First observe that $\Delta(G) \geq 3$, because G is 2-connected, but not a cycle. Let v be a vertex of degree $\Delta(G)$ and set $H := G[N(v) \cup \{v\}]$. We claim that H is not complete. For if $H = G$, then H is not complete by assumption.

Otherwise, $G-H$ is non-empty and by the connectedness of G, there is a vertex $w \in V(H)$ that has a neighbour in $G-H$ and thus satisfies

$$d_H(w) < d_G(w) \leq d_G(v) = \Delta(G) = |H| - 1,$$

implying that H is not complete.

Therefore, H contains two non-adjacent vertices, say a and b. Both of them are neighbours of v and thus a and b have distance two. If $G-a-b$ is connected, we can set $x = a$ and $y = b$ and be done. We may thus assume that $G-a-b$ is not connected. Let C be the component of $G-a-b$ that contains v. Recall that v has degree $\Delta(G) \geq 3$ in G, which implies that v has at least one more neighbour apart from a and b and thus $|C| \geq 2$. If v were the only vertex in C adjacent to a or b, then v would be a cut vertex, contradicting the fact that G is 2-connected. Thus, there is a vertex $x \in C - v$ that is adjacent to a or b, without loss of generality to a.

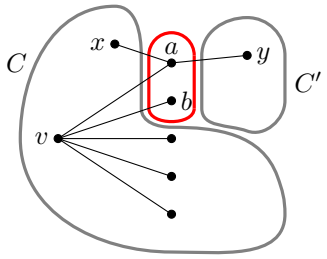

Figure 4.6: Finding the vertices x and y.

As b is not a cut vertex, a has a neighbour in each component of $G-a-b$. Let y be a neighbour of a in some component $C' \neq C$. Then x and y have distance two. To prove that $G-x-y$ is connected, let u be a vertex in $G-x-y$. By Menger's theorem and the 2-connectedness of G, there is a u–$\{a,b\}$ fan in G consisting of two paths. Only one of these paths can meet x or y, because x and y lie in different components of $G-a-b$. Therefore, there is a u–$\{a,b\}$ path—and thus also a u–v path—in $G-x-y$, meaning that $G-x-y$ is connected. □

Proof of Theorem 4.13. Suppose first that G is 2-connected. If G is an even cycle, then $\chi(G) = \Delta(G) = 2$ and we are done. We may thus assume that G is neither complete nor a cycle, hence Lemma 4.14 can be applied. Write $n := |G|$. We denote the two vertices provided by Lemma 4.14 by v_1 and v_2, and let v_n be a common neighbour of them. Perform a depth first search[2] in $G - v_1 - v_2$, starting from v_n, and enumerate the remaining vertices as $v_3, \ldots, v_{n-1}$ so that $\mathrm{DFSnum}[v_i] = n + 1 - i$ (i.e. in the *reverse* order of how DFS discovers them).

Run the greedy algorithm with the ordering $v_1, \ldots, v_n$. Both v_1 and v_2 receive colour 1 by this algorithm. For each i with $3 \leq i \leq n-1$, the parent of v_i in the DFS-tree has larger index and thus does not contribute to the backwards degree of v_i in the greedy algorithm. This means that v_i has backwards degree at most $d(v_i) - 1 \leq \Delta(G) - 1$ and thus receives colour at most $\Delta(G)$ by the algorithm. Finally, v_n has two neighbours with the same colour (v_1 and v_2) and thus at most $\Delta(G) - 1$ colours are used among its neighbours. This implies that

[2] A breadth first search would work as well.

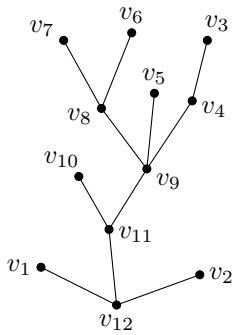

Figure 4.7: A DFS-tree and the corresponding ordering $v_1, \ldots, v_n$ for $n = 12$.

v_n also receives colour at most $\Delta(G)$ and thus the greedy algorithm produces a colouring with at most $\Delta(G)$ colours, as desired.

Now suppose that G is not 2-connected. Let B be a block of G, we claim that it is $\Delta(G)$-colourable. If $\Delta(B) < \Delta(G)$, this follows directly from Proposition 4.8(ii) (applied to B). Otherwise, because B has a vertex v with a neighbour in $G - B$,

$$\delta(B) \leq d_B(v) \leq d_G(v) - 1 \leq \Delta(G) - 1 = \Delta(B) - 1$$

and B is not regular. By Proposition 4.12, B is $\Delta(G)$-colourable. Now Proposition 4.4 proves that $\chi(G) \leq \Delta(G)$. □

4.5 Algorithms for optimal colourings

Naïve approach. The only colouring algorithm we have encountered so far is the greedy algorithm. There is always an ordering of the vertices for which the greedy algorithm produces a colouring of the graph G with $\chi(G)$ colours. A naïve approach would thus be to run the greedy algorithm for all possible orderings and to compare the number of colours used. This would take time

$$\Theta\big((n+m) \cdot n!\big).$$

Zykov's algorithm. We need the following auxiliary result.

Lemma 4.15 (Zykov 1949)**.** *If x, y are non-adjacent vertices of a graph G, then*

$$\chi(G) = \min\Big\{\chi(G + xy), \chi\big(G/\{x,y\}\big)\Big\}.$$

Proof. The colourings of $G + xy$ are precisely those colourings of G for which x and y are coloured differently. Similarly, the colourings of $G/\{x,y\}$ correspond to precisely those colourings of G in which x and y have the same colour (give x and y the colour of $\{x,y\}$, or vice versa). Thus, the desired equality follows from the definition of the chromatic number as the minimum of colours used. □

```
PROCEDURE ZYKOV(G)
BEGIN
   IF (G complete) THEN χ(G) := n;
   ELSE{
      FIND (x, y ∈ V(G) not adjacent);
      ZYKOV(G + xy);
      ZYKOV(G/{x, y});
      χ(G) := min{χ(G + xy), χ(G/{x, y})};
   }
END
```

Let us determine a rough bound for the running time of this algorithm. To this end, we first consider only those calls of the procedure in which the argument is a graph with n vertices. There are $O(n^2)$ many such graphs. Each call of the procedure for such a graph then features a call of a graph with $n-1$ vertices, followed by $O((n-1)^2)$ further calls for graphs with $n-1$ vertices. Recursively, we have to multiply with $O(k^2)$ for the number of calls for graphs with k vertices.[3] Even if we pretend that each call of the procedure only takes constant time, we end up with a running time of

$$O\big((n!)^2\big),$$

which is actually *worse* than the naïve approach. This might partly be due to our rough estimates, but it is not far from the truth.

Proposition 4.16 (McDiarmid 1979)**.** *There exists a constant $\alpha > 0$ such that the Zykov algorithm has running time at least $n^{\alpha n}$ for almost all graphs (that is, for all but a $o(1)$-proportion of all graphs on n vertices).*

Observe that the bound from Proposition 4.16 is much larger than any "simple" exponential bound of the type γ^n.

Remark. *For $k \in \mathbb{N}$ and a graph G, denote by $P_G(k)$ the number of k-colourings of G. An analogous argument to Lemma 4.15 shows that for $e \in E(G)$,*

$$P_G(k) = P_{G-e}(k) - P_{G/e}(k).$$

From this, it is not hard to prove by induction on $\|G\|$ that $P_G(k)$ is a polynomial

$$P_G(k) = k^{|G|} - \|G\|\, k^{|G|-1} + \cdots$$

and is thus called the chromatic polynomial *of G.*

Christofides' algorithm. The next algorithm uses the following easy observation.

Remark. *For every graph G, there is a colouring with $\chi(G)$ colours for which some colour class is a maximal independent set. Thus,*

$$\chi(G) = 1 + \min\{\chi(G-S)\colon S \text{ maximally independent}\}.$$

[3] Each graph on k vertices will be the argument of the procedure multiple times, which is why we have to multiply these numbers instead of summing them. One could change the algorithm so that it is called only once for each graph, but this would make it necessary to save the chromatic number of each graph occurring in the recursion. This would need a memory that is too large to be useful in practice.

We can thus determine the chromatic number recursively, from smaller graphs to larger graphs.

```
BEGIN
    χ(∅) := 0;
    FOR (k = 1 to n){
        FOR (U ∈ (V(G) choose k)){
            χ(G[U]) := k;
            FOR (S max. ind. in G[U]){
                χ(G[U]) := min{χ(G[U]), 1 + χ(G[U \ S])};
            }
        }
    }
END
```

Observe that at the time when $\chi(G[U])$ is computed, the algorithm has already determined the chromatic numbers of all subgraphs $G[U \setminus S]$ of G, because $|U \setminus S| < |U|$.

To estimate the running time of this algorithm, we need to know how many maximally independent sets there are. We use the following result by Christofides without proof.

Lemma 4.17 (Christofides 1971)**.** *For a set U of k vertices in a graph G, there are at most $3^{k/3}$ maximal independent sets in $G[U]$. These sets can be determined in time $O(k^3 3^{k/3})$.*

Using Lemma 4.17, we see that the running time of the algorithm is at most

$$\sum_{k=1}^{n} \binom{n}{k} O(k^3 3^{k/3}) \leq O\left(n^3 \sum_{k=0}^{n} \binom{n}{k} (\sqrt[3]{3})^k\right) = O\left(n^3 (1+\sqrt[3]{3})^n\right).$$

This running time is much faster than for the naïve approach or for Zykov's algorithm. However, Christofides' algorithm needs an exponentially large memory, because all chromatic numbers of subgraphs $G[U]$ have to be remembered. This makes the algorithm hard to use in practice.

Backtracking (Brown 1972). This algorithm colours the vertices one by one similar to the greedy algorithm, thus we need to start with an ordering $v_1, \ldots, v_n$ of the vertices. The main idea is that, when determining the colour of v_k, we try *all* "feasible" colours, including one "new" colour. Moreover, we only continue as long as the number of colours used for $v_1, \ldots, v_k$ is smaller than the current optimum. Otherwise, we go back to the previous vertex.

```
PROCEDURE COLOUR(k)
BEGIN
   NumCols[k] := NumCols[k − 1];
   Queue Available[k] := (1, . . . , NumCols[k]);
   FOR (v_i ∈ N(v_k) ∩ {v_1, . . . , v_{k−1}}){
      Remove c(v_i) from Available[k];
   }
   IF (k < n){
      WHILE (Available[k] ≠ ∅ AND NumCols[k] < χ){
         c(v_k) := First(Available[k]);
         COLOUR(k + 1);
         Remove c(v_k) from Available[k];
      }
      IF (NumCols[k] < χ − 1){
         NumCols[k] := NumCols[k] + 1;
         c(v_k) := NumCols[k];
         COLOUR(k + 1);
      }
   }
   ELSE IF (Available[k] ≠ ∅){
      χ := NumCols[k];
   }
   ELSE{
      χ := min{χ,NumCols[k]+1};
   }
END
```

The main program first has to choose the ordering of the vertices (e.g. via a smallest last algorithm). For the number χ of colours used in the "best" colouring so far, we start with the trivial upper bound n.

```
BEGIN
   χ := n;
   Choose an ordering (v_1, . . . , v_n);
   c(v_1) := 1;
   NumCols[1] := 1;
   COLOUR(2);
   RETURN χ;
END
```

When we call COLOUR(2), we clearly assume that the graph has at least two vertices.

The first colouring that the backtracking algorithm finds is precisely the colouring found by the greedy algorithm with the given ordering of the vertices. All subsequent steps will then try to improve this colouring.

All possible ways to colour G can be encoded in a tree T_G as follows.

- Each vertex in the k-th layer of T_G ($0 \leq k \leq n-1$) represents a partial colouring of G in which precisely $v_1, \ldots, v_{k+1}$ are coloured.
- For $k \geq 1$, each vertex in the k-th layer of T_G is the child of the vertex in the $(k-1)$-th layer that is obtained by removing the colour of v_{k+1}.

The backtracking algorithm goes through the tree in lexicographical order. Whenever it reaches a leaf, the current "optimal" number χ of colours is improved. After that, only branches of the tree that use less than χ colours will be considered by the algorithm.

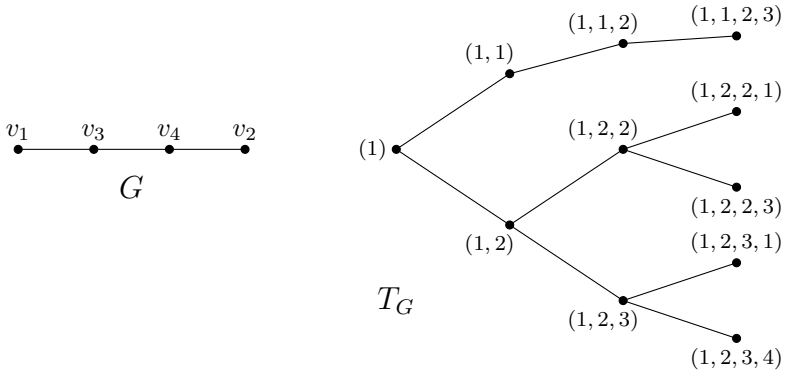

Figure 4.8: The auxiliary tree T_G for a path of length three, where the labels denote the colours of $v_1, \ldots, v_k$ in this order. The backtracking algorithm will first explore the topmost branch in T_G until it reaches the leaf $(1, 1, 2, 3)$. Then it sets $\chi := 3$, which effectively removes the vertices $(1, 2, 2, 3)$, $(1, 2, 3)$, $(1, 2, 3, 1)$, and $(1, 2, 3, 4)$ from the tree. After that, the topmost (in this case only) remaining branch ending in $(1, 2, 2, 1)$ is explored.

Remark. *It has been shown that for "most" graphs, there is an optimal colouring that corresponds to an "early" branch in the auxiliary tree, independently from the choice of the initial ordering $(v_1, \ldots, v_n)$. Most of the time will be spent by the algorithm to check that no "later" branch uses less colours. Therefore, choosing the ordering in a specific way has rather little effect.*

In terms of running time, the worst-case bound for the backtracking algorithm is $O(n!)$, but there exists a constant $\gamma > 1$ such that for almost all graphs, the running time is actually $O(\gamma^n)$. This means that while backtracking is theoretically slow, it is nevertheless reasonably fast in practice.

Remark. *Identifying 2-colourable (i.e. bipartite) graphs is easy (exercise). For fixed $k \geq 3$, determining whether a given graph is k-colourable is NP-complete. The problem of computing the chromatic number is NP-hard.*

4.6 Colouring heuristics

In this section, we will see some algorithms that do not necessarily find an optimal colouring, but that are efficient and produce an optimal colouring in "many" cases. As a minimum requirement, we would like these algorithms to be able to identify bipartite graphs, that is, they should find a 2-colouring whenever the input graph is bipartite.

Saturation-largest-first heuristic (Brélaz 1979). The main idea of this algorithm is to colour the vertices greedily, but to choose the ordering dynamically. While the smallest-last ordering always put the vertex with *smallest*

degree to the *last* position, we will now always choose a vertex that has *as many different colours among its neighbours as possible* and put it at the *first* position.

Definition (Saturation-largest-first algorithm)**.** If some vertices of a graph G are coloured by a partial colouring c, we define the *forbidden colours* of a vertex v as

$$N_c(v) := \{c(u) \colon u \text{ is a coloured neighbour of } v\}.$$

The *saturation-largest-first algorithm* is defined by recursively setting

$$v_i := \operatorname{argmax}\{|N_c(v)| : v \in V(G) \setminus \{v_1, \dots, v_{i-1}\}\}$$

and

$$c(v_i) := \min(\{1, \dots, i\} \setminus N_c(v)).$$

Proposition 4.18. *The saturation-largest-first algorithm colours all bipartite graphs optimally.*

Proof. Suppose first that G is connected and has at least one edge; then its sides A, B are non-empty and uniquely determined (up to renaming). The first vertex v_1 is chosen arbitrarily, say from side A, and receives colour 1. From that time on, every vertex v_i will have neighbours of exactly one colour—neighbours of colour 1 if $v_i \in B$ and neighbours of colour 2 if $v_i \in A$. Thus, all of A will eventually receive colour 1, while B will be coloured with colour 2.

If G is not connected, then the saturation-largest-first algorithm will first colour all vertices of some component before going to the next component. Every isolated vertex will be coloured with colour 1, while every larger component needs two colours as we have seen above. □

Remark. *The saturation-largest-first algorithm can be implemented to run in time $\Theta(n + m)$. A variant of this algorithm colours, for every fixed $\varepsilon \in (0, 1)$, almost all graphs G with $\chi(G) \leq (1 - \varepsilon) \log_2 |G|$ optimally. This variant has running time $\Theta(m \log_2 n)$ (provided that $m = \Omega(n)$).*

Similarity-merge heuristic (Dutton, Brigham 1981). If two vertices in a graph have the same neighbourhood, then they can be coloured with the same colour in any optimal colouring. The rough idea of the next algorithm is to look for non-adjacent vertices with as many common neighbours as possible and to contract them, like in Zykov's algorithm (but without adding edges).

Definition (Similarity-merge algorithm)**.** Set $H_0 := G$. Recursively for $i = 0, 1, \dots,$ repeat the following steps.

(i) If H_i is complete, stop the recursion and set $k := |H_i| = |G| - i$.

(ii) Otherwise, let x_i, y_i be a pair of non-adjacent vertices of H_i with the most common neighbours.

(iii) Set $H_{i+1} := H_i / \{x_i, y_i\}$.

The algorithm then produces a k-colouring by first colouring the last iteration $H_{|G|-k}$ with k colours and then, recursively for $i = |G| - k, |G| - k - 1, \dots, 1$, colour H_{i-1} by letting x_i and y_i inherit the colour of $\{x_i, y_i\}$ in H_i, while all other vertices keep their colours.

Proposition 4.19. *The similarity-merge algorithm colours all bipartite graphs optimally.*

Proof. If G has no edges, then the similarity-merge algorithm will clearly only need one colour (or zero colours if the graph is empty). We may thus assume that G has at least one edge and hence $\chi(G) = 2$. We claim that all graphs $H_0, \dots, H_{|G|-k}$ occurring throughout the algorithm are bipartite and have at least one edge; this will imply that $H_{|G|-k} = H_{|G|-2} = K^2$ and thus the algorithm uses two colours.

$H_0 = G$ satisfies these conditions by assumption. Inductively for $i = 0, \dots, |G| - k - 1$, assume that H_i is bipartite and has at least one edge. If all components of H_i are complete, then a pair of vertices in distinct components is contracted, which keeps the graph bipartite with at least one edge. Otherwise, each non-complete component of H_i contains a path of length two whose end vertices are non-adjacent vertices (because otherwise H_i would contain a triangle, contradicting the fact that it is bipartite) with common neighbours. In particular, x_i, y_i will have at least one common neighbour and thus lie on the same side of their component. Thus, contracting $\{x_i, y_i\}$ keeps the graph bipartite with at least one edge. □

Remark. *The similarity-merge algorithm can be implemented to run in time $\Theta(n^2)$. A variant of this algorithm colours (with longer, but still polynomial running time) almost all graphs G with $\chi(G) \leq \frac{1}{14}\sqrt{\frac{|G|}{\log_2|G|}}$ optimally.*

Recursive-largest-first heuristic. The main intuition behind the third and final heuristic in this section is to look for an independent set S in G (which will be our first colour class) with as many incident edges as possible. This way, we decrease the number of edges in $G - S$ as much as possible, which makes it reasonable to hope that the remaining graph will become edgeless after "few" steps.

Definition (Recursive-largest-first algorithm)**.** Set $H_0 := G$. Recursively for $i = 0, 1, \dots,$ repeat the following steps.

(i) If H_i is empty, stop the recursion and set $k := i$.

(ii) Otherwise, recursively find a maximal set $S_i = \{v_i^1, \dots, v_i^{s_i}\}$ of vertices of H_i such that each v_i^j

  (a) is not adjacent to any v_i^l with $l < j$;

  (b) among all vertices satisfying (a), has the largest number of neighbours in $N_{H_i}(\{v_i^1, \dots, v_i^{j-1}\})$;

  (c) among all vertices satisfying both (a) and (b), has the smallest degree in H_i.

(iii) Set $H_{i+1} := H_i - S_i$.

The algorithm then produces a k-colouring with colour classes $S_0, \dots, S_{k-1}$.

In Step (ii) above, Condition (a) guarantees that S_i is independent. Conditions (b) and (c) are supposed to guarantee that we end up with "many" vertices in S_i that send "many" edges to $H_i - S_i$. For the first of these two

aims, it seems reasonable to pick a vertex with as *few* "new" neighbours (i.e. neighbours of v_i^j that are not neighbours of any earlier v_i^l) as possible. For the second aim, however, we want v_i^j to have as *many* neighbours as possible. These antipodal aims are the reason why we (b) first restrict ourselves to vertices with a maximal number of "already known" neighbours and (c) among those choose a vertex with as few "new" neighbours as possible.

Proposition 4.20. *The recursive-largest-first algorithm colours all bipartite graphs optimally and can be implemented to run in time $O(nm)$.*

Proof. Exercise. □

Remark. *In empirical tests, the recursive-largest-first algorithm needed (on average) the least number of colours of the heuristics in this section.*

4.7 Edge-colouring

Definition (Edge-colouring, chromatic index)**.** An *edge-colouring* of a graph G is a function $c\colon E(G) \to S$ that satisfies $c(e) \neq c(e')$ for all distinct edges with $e \cap e' \neq \emptyset$. (In words: adjacent edges have different colours.)

Analogously to (vertex) colourings, we define k-edge-colourings and k-edge-colourability. The *chromatic index* $\chi'(G)$ of G is defined as

$$\chi'(G) := \min\{k \in \mathbb{N}\colon G \text{ is } k\text{-edge-colourable}\}.$$

Instead of colouring the edges of G, we can also colour the vertices of an auxiliary graph.

Definition (Line graph)**.** The *line graph* of a graph G is the graph $L(G)$ with vertex set $E(G)$, in which distinct $e, e' \in E(G)$ are adjacent if and only if they intersect.

Every k-colouring of $L(G)$ corresponds to a k-edge-colouring of G and vice versa. In particular, $\chi'(G) = \chi(L(G))$. This fact can be used to prove upper and lower bounds for the chromatic index that only differ by a constant factor (as opposed to the bounds we saw for the chromatic number).

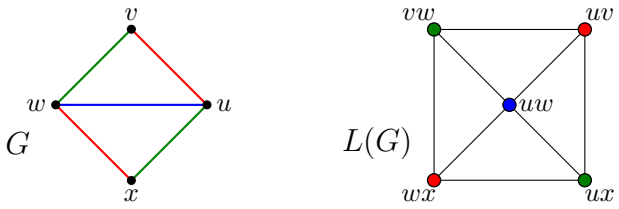

Figure 4.9: A k-edge-colouring of a graph G and the corresponding k-colouring of its line graph $L(G)$.

Proposition 4.21. *For every graph G with $\|G\| \geq 1$, we have $\Delta(G) \leq \chi'(G) \leq 2\Delta(G) - 1$.*

Proof. The set of edges that are incident with a fixed vertex $v \in V(G)$ forms a clique in $L(G)$. Thus, by Proposition 4.1(i),

$$\Delta(G) \leq \omega(L(G)) \leq \chi(L(G)) = \chi'(G).$$

For the upper bound, observe that $e = uv \in E(G)$ has $(d(u)-1)+(d(v)-1)$ many neighbours in $L(G)$ and thus $\Delta(L(G)) \leq 2\Delta(G) - 2$. By Proposition 4.8(ii), this yields $\chi'(G) = \chi(L(G)) \leq 2\Delta(G) - 1$. □

Although Proposition 4.21 already gives us much better control over $\chi'(G)$ than what we know for $\chi(G)$, we shall soon see that it is far from best possible. But first, let us show that for bipartite graphs, the chromatic index lies at the lower end of the possible spectrum.

Theorem 4.22 (König 1916)**.** *If G is bipartite, then $\chi'(G) = \Delta(G)$.*

Proof. Induction on $\|G\|$. An edgeless graph is 0-edge-colourable, we may thus assume that $\|G\| \geq 1$ and that $\chi'(G') = \Delta(G')$ for all bipartite graphs G' with less than $\|G\|$ edges. By Proposition 4.21, it suffices to prove that G is $\Delta(G)$-edge-colourable. Pick any edge $e = xy$ of G. By the induction hypothesis, $G-e$ is $\Delta(G-e)$-edge-colourable and thus in particular $\Delta(G)$-edge-colourable; fix a $\Delta(G)$-edge-colouring c of $G - e$.

Both x and y have at most $\Delta(G) - 1$ neighbours in $G - e$ and thus there are colours $\alpha, \beta \in \{1, \ldots, \Delta(G)\}$ such that α is not used for the edges at x, while β is not used for the edges at y. If α is also not used for the edges at y, then we can colour e with α and be done. We may thus assume that α is used for some edge at y, and analogously that β is used for some edge at x; in particular, $\alpha \neq \beta$. Consider the spanning subgraph $G_{\alpha,\beta}$ of G whose edges are precisely the edges coloured α and β. As c is an edge-colouring, every vertex in $G_{\alpha,\beta}$ has degree zero, one, or two, meaning that each component is either a path or a cycle.

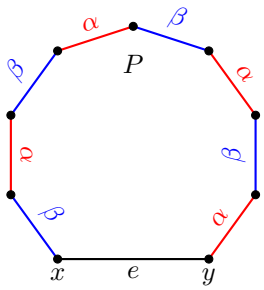

Figure 4.10: Finding an odd cycle if x and y lie in the same component P of $G_{\alpha,\beta}$.

Both x and y have degree one in $G_{\alpha,\beta}$ and thus lie in components that are paths of positive length, are they are endpoints of these paths. Suppose first that they lie in the same component P. Its edge at x is coloured with β, its edge at y is coloured with α, and the colours alternate on P. This means that P has even length. But then $P+e$ is an odd cycle in G, a contradiction to the fact that G is bipartite. Thus, x and y lie in different components of $G_{\alpha,\beta}$, say in P_x and

P_y, respectively. If we switch colours α and β in P_x, then β is no longer used for the edges at x, and we can colour e with β to create a $\Delta(G)$-edge-colouring of G. □

Bipartite graphs are not the only graphs with $\chi'(G) = \Delta(G)$. For instance, denote by W_k the *wheel* of length $k \geq 3$, that is, a cycle of length k with an additional vertex that is adjacent to all vertices on the cycle. Then $\Delta(W_k) = \chi'(W_k) = k$.

Examples with $\chi'(G) > \Delta(G)$ are again odd cycles, for which we have $\chi'(G) = \Delta(G) + 1$. The next theorem tells us that this is in fact the largest chromatic index that can occur!

Theorem 4.23 (Vizing 1964)**.** $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$ *for every graph* G*.*

Proof. Induction on $\|G\|$. Again, observe that the induction base $\|G\| = 0$ is trivial and that it suffices to prove that G is $(\Delta(G) + 1)$-edge-colourable. For simplicity, let us use the short notations

- $\Delta := \Delta(G)$;
- *colouring* for a $(\Delta + 1)$-edge-colouring;
- if a colouring is given, α-β-*path* for a path all whose edges have colours α and β (where $\alpha \neq \beta$ are fixed colours);
- if a colouring is given, we say that colour α *is missing at the vertex* x if no edge incident with x has colour α. (Observe that at every vertex, there is a missing colour, since we use $\Delta + 1$ colours.)

Like in the proof of Theorem 4.22, we know that for every $e = xy \in E(G)$, the graph $G - e$ has a colouring. We may assume the following.

> For every $e = xy \in E(G)$ and any colouring of $G - e$, if colour α is missing at x and colour β is missing at y, then $\alpha \neq \beta$ and the unique maximal α-β-path starting at x ends in y. $(*)$

Indeed, if $\alpha = \beta$ or if the α-β-path starting in x did not end in y, we could find a colouring of G and be done like in the proof of Theorem 4.22.

Fix an edge xy_0 and let c_0 be a colouring of $G - xy_0$. Let α be a colour missing at x and let β_0 be a colour missing at y_0. By $(*)$, β_0 is *not* missing at x and thus, there is an edge xy_1 with $c_0(xy_1) = \beta_0$. Let β_1 be a colour missing at y_1. If β_1 is also missing at x, we can re-colour xy_1 with colour β_1 and obtain a colouring of $G - xy_0$ in which β_0 is missing at *both* x and y_0, a contradiction to $(*)$. We may thus assume that β_1 is *not* missing at x.

We now let xy_2 be an edge with $c_0(xy_2) = \beta_1$, let β_2 be a colour missing at y_2, and continue until for some k either

- β_k is missing at x, or
- $\beta_k = \beta_i$ for some $i < k$.

Observe that $c_k \colon E(G - xy_k) \to \{1, \ldots, \Delta + 1\}$ defined by

$$c_k(e) := \begin{cases} \beta_j & \text{if } e = xy_j \text{ for } 0 \leq j \leq k-1, \\ c_0(e) & \text{otherwise} \end{cases}$$

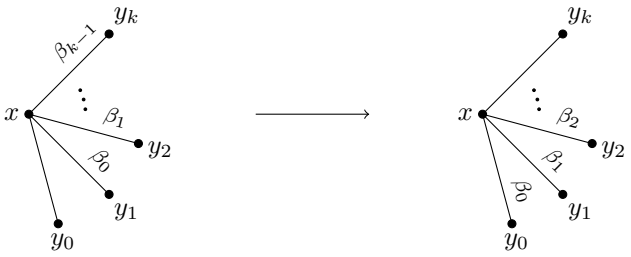

Figure 4.11: Altering c_0 to a colouring of $G - xy_k$.

is a colouring of $G - xy_k$. If β_k is missing at x in c_0, then β_k is missing in c_k at both x and y_0, a contradiction to $(*)$.

We may thus assume that β_k is *not* missing at x, which means that $\beta_k = \beta_i$ for some $i < k$. If we apply $(*)$ to the graph $G - xy_k$ and its colouring c_k, we see that the α-β_i-path P (with respect to c_k) that starts in x ends in y_k. The first edge of P has colour β_i and thus is xy_i, because α is missing at x (in c_0 and hence also in c_k) and $c_k(xy_i) = c_0(xy_{i+1}) = \beta_i$. Analogously, the last edge of P has colour α. This means that y_iP is a y_i–y_k path in $G - \{xy_0, \ldots, xy_k\}$ of odd length.

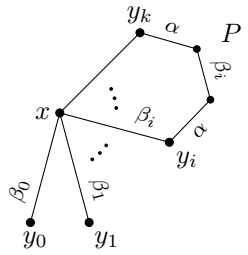

Figure 4.12: Finding the path P using the colouring c_k.

Analogously to c_k, define a colouring c_i of $G - xy_i$ by

$$c_i(e) := \begin{cases} \beta_j & \text{if } e = xy_j \text{ for } 0 \leq j \leq i-1, \\ c_0(e) & \text{otherwise} \end{cases}$$

and switch the colours α and β_i on y_iP to obtain a colouring c_i' of $G - xy_i$ in which α is missing at both x and y_i, contradicting $(*)$. □

Remark. *Vizing's theorem tells us that every graph G lies in one of only two classes of graphs—the ones with $\chi'(G) = \Delta(G)$ (called* Class 1*) or the ones with $\chi'(G) = \Delta(G) + 1$ (called* Class 2*). There are only few further results about which graphs lie in which class. The general problem of determining if a given graph lies in Class 1 or in Class 2 is NP-hard.*

Example. The chromatic index of K^n depends on the parity of n. The cases $n = 1$ and $n = 2$ are trivial. For $n \geq 3$ odd, every colour class can consist of no more than $\frac{n-1}{2}$ edges, showing that one needs n colours to colour all $\binom{n}{2}$ edges.

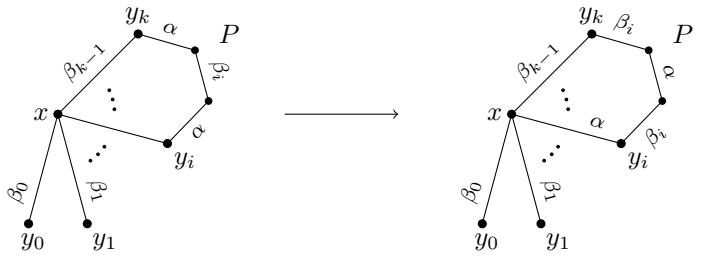

Figure 4.13: Creating a colouring of G from c_i.

Since $n = \Delta(K^n) + 1$, Vizing's theorem proves that $\chi'(K^n) = n$ for n odd. Moreover, we have shown that in every n-edge-colouring of K^n, each colour is used for exactly $\frac{n-1}{2}$ edges. Thus, each colour is missing at precisely one vertex.

For $n \geq 4$ even, delete a vertex x from K^n to obtain a K^{n-1}. Colour its edges with colours $1, \ldots, n-1$ as above and suppose that each colour i is missing at v_i. Now we can colour each remaining edge xv_i with colour i and thus show $\chi'(K^n) = n - 1 = \Delta(K^n)$.

4.8 List colouring

Graph colouring occurs in many applications, e.g. in scheduling problems, where each colour class corresponds to a time slot. In many cases, there are further restrictions like that every vertex/edge has some colours that are forbidden a priori (in the sense that e.g. the event corresponding to a specific vertex/edge has to take place within a certain time range).

Definition (List colouring, choice number)**.** Let G be a graph and suppose that for each vertex $v \in V(G)$, we are given a non-empty set S_v. We say that a (vertex-)colouring c of G is a colouring *from the lists* S_v if $c(v) \in S_v$ for every vertex v.

For $k \in \mathbb{N}$, we say that G is *k-choosable* (or *k-list-colourable*) if, for *every* family of lists of size (at least) k, G has a colouring from these lists. The *choice number* (or *list-chromatic number*) $\mathrm{ch}(G)$ of G is defined as

$$\mathrm{ch}(G) := \min\{k \in \mathbb{N} \colon G \text{ is } k\text{-choosable}\}.$$

The corresponding notions for edge-colourings are defined analogously. The smallest integer k for which G is k-edge-choosable is called the *choice index* (or *list-chromatic index*) of G and is denoted by $\mathrm{ch}'(G)$.

Remark. *A k-colouring can be viewed as a colouring from identical lists of size k. Therefore, every k-choosable graph is in particular k-colourable, which means that $\chi(G) \leq \mathrm{ch}(G)$. Analogously, we have $\chi'(G) \leq \mathrm{ch}'(G)$.*

Example. Consider the graph $K_{3,3}$. On both sides, let one vertex have the list $\{1, 2\}$, one vertex list $\{1, 3\}$, and one vertex list $\{2, 3\}$ (see Figure 4.14).

The vertices in A cannot have the same colour, since the three lists have empty intersection. No matter which colours are chosen for the vertices in A,

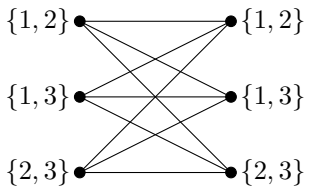

Figure 4.14: An assignment of lists to the vertices of $K_{3,3}$ that does not allow a colouring.

some vertex in B will have both colours in its list already used for the vertices in A. Therefore, $K_{3,3}$ is *not* 2-choosable.

If we have lists of size 3 instead, then either two lists of vertices in A intersect or no such lists intersect. In the former case, we can colour the two corresponding vertices with their common colour; for the third vertex in A use any colour from its list. This way, we have only used two colours for the vertices in A, meaning that each vertex in B still has one colour in its list that we can colour it with. Thus, we find a colouring of $K_{3,3}$ from the given lists.

Otherwise, the lists of vertices in A are pairwise disjoint. We have 27 different possibilities to colour the vertices in A, with the set of colours being different in all 27 cases. Each vertex in B has a list of size three, and so at most one of the 27 colourings will exhaust the list for any vertex. In particular, for at least 24 of these colourings we can extend them to a colouring of the whole of $K_{3,3}$ from the given lists.

This shows that $\mathrm{ch}(K_{3,3}) = 3$.

Some results about bounds on the chromatic number immediately extend to the choice number, with similar proofs.

Proposition 4.24. *For every graph G, we have* $\mathrm{ch}(G) \leq 1 + \max_{H \subseteq G} \delta(H)$.

Theorem 4.25. *If G is connected and neither complete nor an odd cycle, then* $\mathrm{ch}(G) \leq \Delta(G)$.

Proposition 4.24 and Theorem 4.25 are the counterparts of Corollary 4.11(i) and Theorem 4.13 (Brooks' theorem) for the choice number. Their proofs are left as an exercise.

While these results seem to indicate similarities between the chromatic number and the choice number, the two values can lie arbitrarily far apart.

Theorem 4.26 (Alon 1993)**.** *There exists a function $f\colon \mathbb{N} \to \mathbb{N}$ such that for all $k \in \mathbb{N}$ and all graphs G with average degree $d(G) \geq f(k)$, we have* $\mathrm{ch}(G) \geq k$.

Observe that Theorem 4.26 implies that $\mathrm{ch}(K_{f(k),f(k)}) \geq k$, hence there exist bipartite graphs with arbitrarily large choice number![4] The proof of Theorem 4.26 uses probabilistic arguments. We omit the proof in this course.

There is another result that we have already seen for the chromatic number, that still holds for the choice number: the Five Colour Theorem.

Theorem 4.27 (Thomassen 1994)**.** *Every planar graph is 5-choosable.*

[4] One can also directly construct bipartite graphs with arbitrarily large choice number (exercise).

Proof. We prove the following stronger statement.

> If G is a plane graph whose outer face is bounded by a cycle $C = v_1v_2 \dots v_kv_1$ and whose inner faces are bounded by triangles, then for all lists with $S_{v_1} = \{1\}$, $S_{v_2} = \{2\}$, $|S_{v_i}| \geq 3$ for $3 \leq i \leq k$, and $|S_v| \geq 5$ for $v \in V(G - C)$, there is a colouring of G from these lists. (∗)

First observe that (∗) indeed implies the theorem. For if G is maximally planar with at least three vertices,[5] then any drawing of G is a triangulation by Proposition 3.4. In particular, G has the structure required in (∗). If we delete all but one colour each from the lists S_{v_1} and S_{v_2}, then (∗) states that G is colourable from these lists, and thus also from the original lists.

We prove (∗) by induction on $|G|$. The case $|G| = 3$ is trivial, because then G is a triangle and we can colour $c(v_1) = 1$, $c(v_2) = 2$, and pick $c(v_3)$ from $S_{v_3} \setminus \{1, 2\} \neq \emptyset$. Now suppose that $|G| \geq 4$ and that (∗) holds for all smaller plane graphs.

Suppose that there is an index $2 \leq i \leq k - 2$ such that $v_iv_k \in E(G)$. Then $\{v_i, v_k\}$ separates G into two parts; let G_1 be the part that contains v_1, v_2, and let G_2 be the other part. The induction hypothesis for G_1 yields a colouring c_1 of G_1. In G_2, reduce the lists S_{v_i} and S_{v_k} to $\{c_1(v_i)\}$ and $\{c_1(v_k)\}$, respectively, and apply the induction hypothesis to G_2 with v_i and v_k in the roles of v_1 and v_2. Together, c_1 and c_2 form a colouring of G from the original lists.

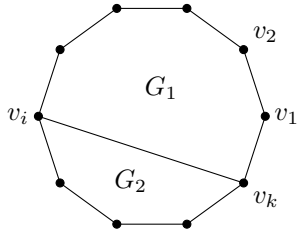

Figure 4.15: If there is an index $2 \leq i \leq k - 2$ with $v_iv_k \in E(G)$, then we can inductively first colour G_1 from the lists and then colour G_2 from the lists so that the colours at c_i and c_k coincide.

Now suppose that there is no such index i. Then we can denote the neighbours of v_k in $V(G - C)$ by $u_1, \dots, u_j$ so that $v_1, u_1, \dots, u_j, v_{k-1}$ lie around v_k in that order. Since every inner face of G is bounded by a triangle, $P := v_1u_1 \dots u_jv_{k-1}$ is a path in G and $C' := (C \cup P) - v_k$ is a cycle in G. Pick any two colours α, β from $S_{v_k} \setminus \{1\}$ (this is possible as S_{v_k} has at least three elements). Delete α and β from each list S_{u_l} with $1 \leq l \leq j$, this leaves at least three elements in each such list. This means that (∗) is applicable to $G - v_k$; let c be the colouring provided by (∗). Amongst the neighbours of v_k, we know that neither v_1 nor any u_l can be coloured with α or β. Thus, we can pick any colour in $\{\alpha, \beta\} \setminus \{c(v_{k-1})\} \neq \emptyset$ for v_k so as to extend c to a colouring of G from the original lists. □

[5] The theorem is trivial for graphs with fewer vertices, and if the theorem is true for all maximally planar graphs, then it is automatically true for all planar graphs.

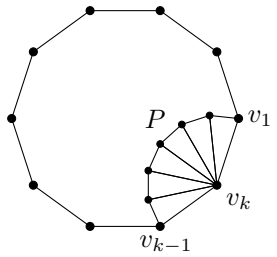

Figure 4.16: If v_k is not adjacent to any v_i with $2 \leq i \leq k-2$, then its neighbours form a v_1–v_{k-1} path P. Colour $G - v_k$ from the lists, not using two fixed colours $\alpha, \beta \in S_{v_k} \setminus \{1\}$ for the internal vertices of P. Then at least one of α, β is available for v_k.

For 4-choosability, the above proof fails (where?). In fact, there exist planar graphs with choice number 5, the smallest known example having 63 vertices.

Recall that the chromatic number and the choice number can be arbitrarily far apart by Theorem 4.26. It might be surprising that not a single example is known for which the chromatic index and the choice index are *not identical*! It is indeed conjectured that these two values are always the same.

List Colouring Conjecture. Every graph G satisfies $\chi'(G) = \mathrm{ch}'(G)$.

The List Colouring Conjecture is still wide open. For bipartite graphs, the conjecture is known to be true.

Theorem 4.28 (Galvin 1995)**.** *If G is bipartite, then $\chi'(G) = \mathrm{ch}'(G)$.*

Proof. Recall that $\chi'(G) = \Delta(G) =: \Delta$ by Theorem 4.22; fix a Δ-edge-colouring c of G. We need to prove that G is Δ-edge-choosable. To that end, suppose that lists S_e of size Δ are given.

We define a directed auxiliary graph D with vertex set $E(G)$ as follows. Let ee' be an edge of the line graph of G and suppose that $c(e) < c(e')$. We let $(e, e') \in E(D)$ if e and e' meet in A, and $(e', e) \in E(D)$ if they meet in B.

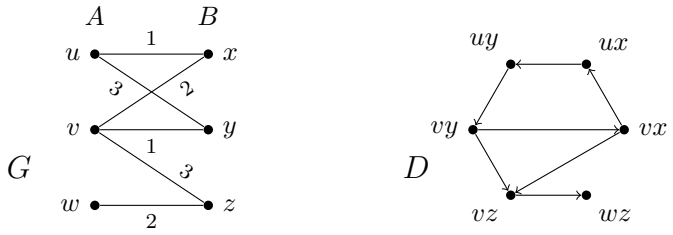

Figure 4.17: Constructing the directed graph D from the graph G.

We claim that D has the following property.

> For every $e \in V(D)$, we have $d^+(e) \leq |S_e| - 1$. Furthermore, for every induced subgraph $D' \subseteq D$, there is an independent set $U \subseteq V(D')$ such that each vertex in $D' - U$ sends an edge to U. $(*)$

If $e = ab \in E(G)$ is fixed (with $a \in A$ and $b \in B$), then $(e, e') \in E(D)$ implies that either $e \cap e' = \{a\}$ and $c(e) < c(e')$ (for which there are $\Delta - c(e)$ possibilities) or $e \cap e' = \{b\}$ and $c(e) > c(e')$ (with $c(e) - 1$ possibilities). This implies the first part of $(*)$, because $|S_e| = \Delta$.

For the second part, suppose that D' is given and let G' be the spanning subgraph of G with edge set $V(D')$. For every $v \in V(G')$, define an order $<_v$ on $N(v)$ by letting $u <_v w$ whenever $(vw, vu) \in E(D')$. By Theorem 2.4, G' has a stable matching with respect to this set of preferences. Any stable matching corresponds to an independent set U in D' (because U is a matching) such that each vertex in $D' - U$ sends an edge to U (because U is stable). This proves the second part of $(*)$.

We will now prove by induction on $|D|$ that every directed graph satisfying $(*)$ can be coloured from the lists S_e. The induction basis $|D| = 0$ is trivial, so suppose that $|D| \geq 1$. By $(*)$, the lists S_e are non-empty; pick any colour α that occurs in some list. Let D' be the subgraph of D spanned by all $e \in V(D)$ for which $\alpha \in S_e$. Apply $(*)$ and colour all elements of the independent set U with α. Now consider the graph $D'' = D - U$ with lists $S''_e = S_e \setminus \{\alpha\}$; we claim that $(*)$ holds for D''.

The second part of $(*)$ holds for D'', because it holds for D and every induced subgraph of D'' is also an induced subgraph of D. For the first part of $(*)$, let $e \in V(D'')$. If $e \in V(D' - U)$, we have $|S''_e| = |S_e| - 1$ (since we deleted the colour α from this list), but also $d^+_{D''}(e) \leq d^+_D(e) - 1$, because e sends an edge to U by $(*)$. For $e \in V(D - D')$, we have $|S''_e| = |S_e|$ and $d^+_{D''}(e) \leq d^+_D(e)$. In either case, the first part of $(*)$ is true for D'', because it was true for D.

By the induction hypothesis, we can colour D'' from the modified lists. Since this does not use the colour α, this yields a colouring of D from the original lists, which corresponds to an edge-colouring of G from the lists S_e, as desired. $\square$

Chapter 5

Euler tours and Hamilton cycles

Definition (Walks)**.** A sequence $W = v_0, e_0, v_1, e_1, \ldots, v_{k-1}, e_{k-1}, v_k$ of (not necessarily distinct) vertices v_i and edges e_i in a graph or multigraph G is called *walk* in G if $e_i = v_i v_{i+1}$ for each $i = 0, \ldots, k-1$. The walk is *closed* if $v_0 = v_k$. We say that an edge e is *traversed l times by W* if $e = e_i$ for precisely l distinct values of i.

5.1 Euler tours

Definition (Euler walks, Euler tours)**.** An *Euler walk* in a (multi-)graph G is a walk that traverses each edge of G exactly once. A closed Euler walk is called *Euler tour*. If G has an Euler tour, we also call G *Eulerian*.

Theorem 5.1 (Euler 1736, Hierholzer 1873)**.** *A connected (multi-)graph G is Eulerian if and only if all vertices have even degrees. G has an Euler walk if and only if at most two vertices have odd degrees.*

Proof. If G has an Euler tour W, then a vertex v that appears j times in W (where we count $v = v_0 = v_k$ as one occurrence) has degree $2j$. Thus, all vertices have even degrees. Vice versa, suppose that all vertices have even degrees and let $W = v_0, \ldots, v_k$ be a longest walk in G that does not traverse any edge more than once. By the maximality of W, it traverses all edges at v_k. Since $d(v_k)$ is even, this means that W is closed. If W were not an Euler tour, then there would be edges that it does not traverse. As G is connected, this means that there is some edge e that is not traversed by W and that is incident with some vertex of W, say $e = uv_i$. But then

$$W' := u, e, v_i, e_i, v_{i+1}, \ldots, v_{k-1}, e_{k-1}, v_k, e_0, v_1, \ldots, v_i$$

would be a longer walk than W, a contraction. Thus, W is an Euler tour.

If G has an Euler walk W, then either W is closed and hence all vertex degrees are even (as above) or W is not closed and all vertices apart from the end vertices of W have even degrees. Vice versa, suppose that at most two vertex degrees are odd. Then by Proposition 0.1, either all vertices have even

degrees—in which case we have already seen that G has an Euler tour, which is in particular an Euler walk—or precisely two vertices u, v have odd degrees. In this case, add an edge $e = uv$ to G to obtain a multigraph G' (adding e might create a new double edge) in which all vertices have even degrees. Then G' has an Euler tour W. As W is closed, we may assume that it starts with u, e, v. Deleting the first two elements from the sequence results in an Euler walk in G (starting at v and ending at u). □

Clearly, adding isolated vertices to an Eulerian graph does not change anything, as we only ask for all *edges* to be traversed.[1]

By Theorem 5.1, we can easily check in linear time whether a given graph G is Eulerian. To this end, we in particular need to check connectedness (after deleting isolated vertices), which can e.g. be done by depth first search or breadth first search.

```
BEGIN
    Eulerian := TRUE;
    H := G;
    FOR (v ∈ V(G)){
        IF (d(v) is odd){
            Eulerian := FALSE;
        }
        IF (d(v) = 0){
            H := H − v;
        }
    }
    If (Connected(H) = FALSE){
        Eulerian := FALSE;
    }
    RETURN Eulerian;
END
```

We can also find an Euler tour algorithmically, along the lines of the proof of Theorem 5.1. Suppose that we have already checked that G is Eulerian. The main idea is to start with any closed walk W that traverses each edge at most once and as long as this walk is not yet an Euler walk, extend W by adding a closed walk in the remaining graph that starts (and ends) at a vertex that also lies on W. Suppose first that we already know which vertex v to start from.

```
PROCEDURE TOUR(G,v)
BEGIN
    w_0 := v;
    t := 0;
    WHILE (t = 0 OR w_t ≠ v){
        Find w_{t+1} ∈ N(w_t);
        e_t := w_t w_{t+1};
        G := G − e_t;
        t := t + 1;
    }
    T := w_0 e_0 w_1 ... w_{t−1} e_t w_t;
END
```

[1] Some sources might require an Eulerian graph to be connected, but this makes virtually no difference.

The main program will now recursively apply this procedure. For simplicity, assume that a vertex s in G is given from which we start our Euler tour.

```
BEGIN
    R := G;
    W := s;
    WHILE (E(R) ≠ ∅){
        FIND (w ∈ V(W): d_R(w) ≠ 0);
        TOUR(R,w);
        W := sWwTwWs;
        R := R − E(T);
    }
END
```

This algorithm is due to Hierholzer and runs in time $\Theta(n+m)$. A very similar algorithm also works if G has two vertices with odd degrees: we just have to find a path between these vertices first, and then recursively apply Tour(R,w) as above.

Note that, TOUR(G, v) will always find a closed walk as long as all degrees in G are even. Indeed, during the WHILE loop the leading vertex of the path w_t will always have odd degree in the current graph G, since we delete one edge incident to w_t in steps t and $t + 1$. In particular, w_t always has at least one neighbour in G. Since the number of edges in G at the start is finite, the loop must terminate, in which case we must have that $w_t = v$.

5.2 Hamilton cycles

Definition (Hamilton cycles, Hamilton paths)**.** A *Hamilton cycle* in a graph G is a cycle that is a spanning subgraph of G (in other words, a cycle in G that visits all vertices). A graph that has a Hamilton cycle is called *hamiltonian*.

A *Hamilton path* is defined analogously to a Hamilton cycle.

While identifying Eulerian graphs is easy (even possible in linear time), the corresponding problem for hamiltonian graphs is much harder.

Remark. *The problem of determining whether a given graph G is hamiltonian is NP-complete.*

5.2.1 Sufficient conditions for Hamilton cycles

What properties can guarantee hamiltonicity of a graph? Large degrees might be a sensible candidate, but any constant minimum degree will not even guarantee connectedness, let alone a Hamilton cycle. Thus, any such sufficient condition has to feature a lower bound that depends on the size of the graph.

Theorem 5.2 (Dirac 1952)**.** *Every graph G with $|G| \geq 3$ and $\delta(G) \geq \frac{1}{2}|G|$ is hamiltonian.*

Proof. Observe that G is connected, for otherwise there would be a component with at most $\frac{1}{2}|G|$ vertices and thus $\delta(G) < \frac{1}{2}|G|$.

Let $P = v_0 \dots v_k$ be a longest path in G. We first aim to find a cycle C in G with the same vertex set as P. By the maximality of P, we have $N(v_0) \subseteq \{v_1, \dots, v_k\}$ and $N(v_k) \subseteq \{v_0, \dots, v_{k-1}\}$. Let

$$S := \{v_i \colon v_{i+1} \in N(v_0)\}$$

be the set of predecessors (on P) of neighbours of v_0. We know that both $N(v_k)$ and S are contained in $\{v_0, \dots, v_{k-1}\}$ and that $|N(v_k)| \geq \frac{1}{2}|G|$ and $|S| = |N(v_0)| \geq \frac{1}{2}|G|$. Since $|\{v_0, \dots, v_{k-1}\}| = k < |G|$, the pigeonhole principle yields that $N(v_k) \cap S \neq \emptyset$. Let $v_i \in N(v_k) \cap S$. Then

$$C = (P - v_i v_{i+1}) + v_0 v_{i+1} + v_i v_k$$

is the desired cycle.

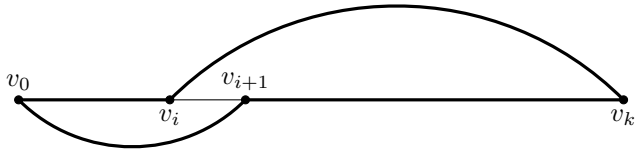

Figure 5.1: Finding a cycle with the same vertex set as the longest path in G.

If $V(C) \neq V(G)$, then there exists a vertex $x \in V(G) - V(C)$ that is adjacent to some vertex $y \in V(C)$, because G is connected. Adding xy to C and deleting one of the two edges of C incident with y creates a path that is longer than P, a contradiction. Therefore, C is a spanning subgraph of G and thus a Hamilton cycle. □

Alternative proof. Among all orderings of $V(G)$, pick $\sigma_0 = (v_1, \dots, v_n)$ with

$$k(\sigma_0) := |E(G) \cap \{v_1v_2, v_2v_3, \dots, v_{n-1}v_n, v_nv_1\}|$$

maximal. We aim to prove that $k(\sigma_0) = n$; then $v_1v_2 \dots v_nv_1$ is a Hamilton cycle and we are done.

Suppose, for contradiction, that $k(\sigma_0) < n$. Without loss of generality, we may assume that $v_nv_1 \notin E(G)$. Let $S := \{v_i \colon v_{i+1} \in N(v_1)\}$. Neither S nor $N(v_n)$ contains v_n and thus, the pigeonhole principle implies that there exists some $v_i \in N(v_n) \cap S$, because $|S|, |N(v_n)| \geq \frac{1}{2}n$. Also observe that $2 \leq i \leq n-2$, because $v_1 \notin N(v_n)$, while $v_{n-1} \notin S$. Consider the ordering $\sigma_1 := (v_1, \dots, v_i, v_n, v_{n-1}, \dots, v_{i+1})$.

Among all edges counted in $k(\sigma_0)$, we might only lose v_iv_{i+1} (if this was an edge) for $k(\sigma_1)$. On the other hand, we gain the edges $v_{i+1}v_1$ and v_iv_n. Thus, $k(\sigma_1) > k(\sigma_0)$, a contradiction to the choice of σ_0. □

Remark. *Dirac's theorem is best possible in the sense that any smaller (integral) minimum degree does not suffice. For any integers d, n with $1 \leq d < \frac{1}{2}n$, the complete bipartite graph $K_{d,n-d}$ is an example of a graph with minimum degree d and no Hamilton cycle. A different example would be the union of two complete graphs K^{d+1} and K^{n-d} that meet in a single vertex.*

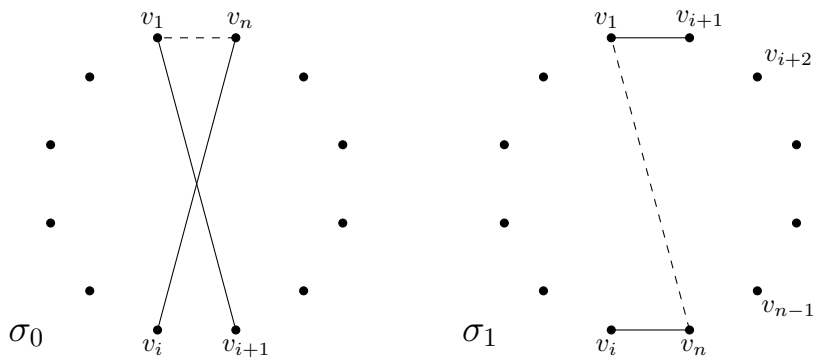

Figure 5.2: Constructing σ_1 from σ_0.

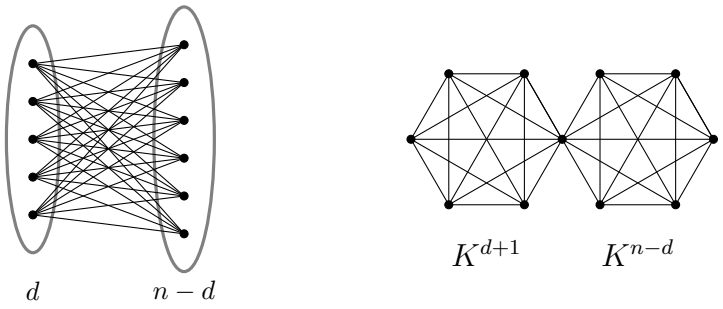

Figure 5.3: Two graphs on $n = 11$ vertices with minimum degree $d = 5 < \frac{1}{2}n$, but no Hamilton cycle (why?).

The degree condition in Dirac's theorem can in fact be relaxed. Instead of asking each vertex to have large degree, it is enough to require pairs of vertices to have large sums of their degrees.

Theorem 5.3 (Ore 1960)**.** *Let G be a graph with $|G| \geq 3$ and suppose that $d(x) + d(y) \geq |G|$ for all distinct, non-adjacent vertices x, y of G. Then G is hamiltonian.*

Proof. Exercise. □

Remark. *Every graph that satisfies the conditions of Dirac's theorem also satisfies the conditions of Ore's theorem. Thus, Theorem 5.3 implies Theorem 5.2.*

Theorem 5.3 is again best possible: The second example in Figure 5.3 shows that there are non-hamiltonian graphs G with $d(x) + d(y) = |G| - 1$ for any two non-adjacent vertices x, y. Observe that said example is not hamiltonian for any *d from 1 up to $n - 2$. (For $d = 1$ or $d = n - 2$, the example reduces to a complete graph on $n - 1$ vertices plus one last vertex that has degree one.)*

Another obvious candidate for ensuring hamiltonicity is large connectivity. Dirac's theorem, together with the trivial fact that $\delta(G) \geq \kappa(G)$, tells us that every $\lceil \frac{1}{2}n \rceil$-connected graph G on n vertices is hamiltonian. Again, this bound is best possible, as shown by the first example in Figure 5.3 for $d = \lceil \frac{1}{2}n \rceil - 1$.

Interestingly, if we compare the connectivity of G with another graph constant, we find a surprising further sufficient condition for a Hamilton cycle.

Theorem 5.4 (Chvátal, Erdős 1972)**.** *Every graph G with at least three vertices and $\kappa(G) \geq \alpha(G)$ is hamiltonian.*

Proof. If $\alpha(G) = 1$, then G is complete (with at least three vertices) and hence hamiltonian. We may thus assume that $\kappa(G) \geq \alpha(G) \geq 2$.

Then G contains a cycle by Proposition 1.8. Let C be a longest cycle in G. If C is not a Hamilton cycle, pick a vertex v in $G - C$. By the fan version of Menger's theorem (Theorem 1.7(iii)), the maximal v–$V(C)$ fan consists of $k \geq \kappa(G)$ paths $P_1, \ldots, P_k$. Denote their end vertices on C by $v_1, \ldots, v_k$, respectively. If for some $i \neq j$, v_i and v_j are adjacent on C, then

$$(C \cup P_i \cup P_j) - v_i v_j$$

would be a longer cycle than C, a contradiction. Thus, no two vertices v_i, v_j are adjacent on C.

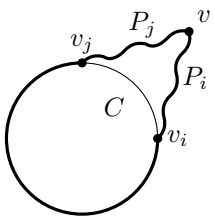

Figure 5.4: Constructing a longer cycle if v_i and v_j are adjacent on C.

For each i, let w_i be the neighbour of v_i on C in some fixed direction. Then $v_1, \ldots, v_k, w_1, \ldots, w_k$ are pairwise distinct. If v were adjacent to any w_i, then

we could increase the v–$V(C)$ fan by adding the edge vw_i as a path. By the maximality of our fan, no such edge exists. But $\{v, w_1, \dots, w_k\}$ cannot be an independent set, because it has size $k+1 > \kappa(G) \geq \alpha(G)$. Thus, some vertices w_i, w_j are adjacent (in G, not in C). Now

$$(C \cup P_i \cup P_j) + w_i w_j - v_i w_i - v_j w_j$$

is a longer cycle than C, again a contradiction. □

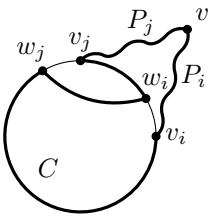

Figure 5.5: Constructing a longer cycle if w_i and w_j are adjacent.

Remark. *Theorem 5.4 is stronger than Theorem 5.3: Every graph G with at least three vertices and $d(x) + d(y) \geq |G|$ for all distinct non-adjacent vertices x, y also satisfies $\kappa(G) \geq \alpha(G)$ (exercise).*

Theorem 5.4 is best possible in the sense that there are non-hamiltonian graphs G with $\alpha(G) = \kappa(G) + 1$, e.g. the complete bipartite graph $K_{r,r+1}$.

Finally let us mention another classical result. To this end, we need the following definition.

Definition (Powers of a graph)**.** For a graph G and a positive integer k, we define the *k-th power* G^k of G as the graph on $V(G)$ with edge set

$$E(G^k) := \{xy \colon x, y \text{ have distance at most } k \text{ in } G\}.$$

Theorem 5.5 (Fleischner 1974)**.** *The square G^2 of a 2-connected graph G is hamiltonian.*

5.2.2 Hamiltonian sequences

Ore's theorem (Theorem 5.3) gives a sufficient condition for hamiltonicity in terms of vertex degrees. But what happens if we only look at the list of degrees in a graph and forget about which vertex had which degree? Will we still be able to find a nice condition, preferably weaker than the condition "all degrees are larger than $\frac{1}{2}|G|$" from Dirac's theorem?

Definition (Degree sequence, hamiltonian sequences)**.** Given a graph G on n vertices, let $v_1, \dots, v_n$ be an ordering of $V(G)$ such that $d(v_1) \leq \dots \leq d(v_n)$. Then we call $d(v_1), \dots, d(v_n)$ the *degree sequence* of G.

A sequence $a_1, \dots, a_n$ of non-negative integers with $a_1 \leq \dots \leq a_n \leq n-1$ is called *hamiltonian* if every graph on n vertices with pointwise larger degree sequence (that is, $d(v_i) \geq a_i$ for each i) is hamiltonian.

The next result characterises hamiltonian sequences.

Theorem 5.6 (Chvátal 1972)**.** *Let $n \geq 3$. A sequence $a_1, \ldots, a_n$ of non-negative integers with $a_1 \leq \cdots \leq a_n \leq n-1$ is hamiltonian if and only if for every $i < \frac{n}{2}$ with $a_i \leq i$, we have $a_{n-i} \geq n - i$.*

Observe that Theorem 5.6 does *not* say anything about the sizes of $a_{n/2}$ (if n is even) and a_n, apart from the condition $0 \leq a_1 \leq \cdots \leq a_n \leq n-1$.

Proof. Suppose first that $a_1, \ldots, a_n$ satisfies the above property. We prove by induction on $\binom{n}{2} - \|G\|$, that every graph G on n vertices with degree sequence pointwise larger than $a_1, \ldots, a_n$ is hamiltonian. The induction base $\|G\| = \binom{n}{2}$ is trivial, because every complete graph on $n \geq 3$ vertices contains a Hamilton cycle.

For the induction step, suppose that G is not complete. Pick non-adjacent vertices x, y with $d(x) \leq d(y)$ and $d(x) + d(y)$ as large as possible. The degree sequence of $G + xy$ is pointwise larger than that of G, and thus also pointwise larger than $a_1, \ldots, a_n$. By the induction hypothesis, $G+xy$ contains a Hamilton cycle C. If $xy \notin E(C)$, then C is a Hamilton cycle of G and we are done. We may thus assume that C contains xy. In other words, $C - xy$ is a Hamilton path in G from x to y, say $C - xy = x_1 \ldots x_n$ (with $x_1 = x$ and $x_n = y$). Like in the proof of Dirac's theorem, we find a Hamilton cycle of G as soon as $N(y)$ and $S := \{x_i \colon x_{i+1} \in N(x)\}$ are not disjoint, which is in particular the case if $d(x) + d(y) \geq n$. We may thus assume that $N(y) \cap S = \emptyset$, which implies

$$d(x) + d(y) < n, \quad \text{hence in particular} \quad d(x) < \frac{n}{2}.$$

Let v be a vertex in S, then it is not adjacent to y. If $d(v) > d(x)$, this would contradict the maximality of $d(x) + d(y)$, hence all vertices in S have degree at most $d(x)$. This means that there are at least $|S| = d(x)$ many vertices in G with degree at most $d(x)$. In other words, if we set $i := d(x)$, then $i < \frac{n}{2}$ and $a_i = d(v_i) \leq i$. This implies $d(v_{n-i}) = a_{n-i} \geq n - i$, i.e. G has at least $i + 1 = d(x) + 1$ vertices of degree at least $n - i = n - d(x)$. At least one such vertex, call it z, is not adjacent to x. But then $d(x) + d(z) \geq n > d(x) + d(y)$, contradicting the choice of x, y. This proves that G has a Hamilton cycle. By induction, $a_1, \ldots, a_n$ is hamiltonian.

Vice versa, suppose that $a_1, \ldots, a_n$ is an integer sequence and that for some $i < \frac{n}{2}$, both $a_i \leq i$ and $a_{n-i} \leq n - i - 1$. Let $H_{n,i}$ be the graph on $\{v_1, \ldots, v_n\}$ that is the (edge-disjoint) union of a K^{n-i} on $\{v_{i+1}, \ldots, v_n\}$ and a $K_{i,i}$ with sides $\{v_1, \ldots, v_i\}$ and $\{v_{n-i+1}, \ldots, v_n\}$.

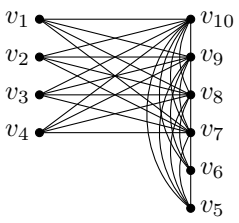

Figure 5.6: The graph $H_{10,4}$.

Observe that

- for $1 \leq j \leq i$, we have $d(v_j) = i \geq a_i \geq a_j$;
- for $i+1 \leq j \leq n-i$, we have $d(v_j) = n-i-1 \geq a_{n-i} \geq a_j$;
- for $n-i+1 \leq j \leq n$, we have $d(v_j) = n-1 \geq a_n \geq a_j$.

In other words, the degree sequence

$$\underbrace{i, \ldots, i}_{i \text{ times}}, \underbrace{n-i-1, \ldots, n-i-1}_{n-2i \text{ times}}, \underbrace{n-1, \ldots, n-1}_{i \text{ times}}$$

of $H_{n,i}$ is pointwise larger than $a_1, \ldots, a_n$. But $H_{n,i}$ is not hamiltonian, because the set $\{v_{n-i+1}, \ldots, v_n\}$ of i vertices separates $H_{n,i}$ into $i+1$ components (the isolated vertices $v_1, \ldots, v_i$ and a complete graph on $\{v_{i+1}, \ldots, v_{n-i}\}$). This proves that $a_1, \ldots, a_n$ is not hamiltonian. □

Instead of Hamilton cycles, we could also ask for Hamilton paths.

Definition (Path-hamiltonian sequences)**.** Call a sequence $a_1, \ldots, a_n$ of non-negative integers with $a_1 \leq \cdots \leq a_n \leq n-1$ *path-hamiltonian* if every graph on n vertices with pointwise larger degree sequence has a Hamilton path.

Corollary 5.7. *For every $n \geq 1$, a sequence $a_1, \ldots, a_n$ of non-negative integers with $a_1 \leq \cdots \leq a_n \leq n-1$ is path-hamiltonian if and only if for every $i \leq \frac{n}{2}$ with $a_i < i$, we have $a_{n+1-i} \geq n-i$.*

Proof. The claim is trivial for $n = 1$, because the only feasible sequence in that case is $a_1 = 0$, which is path-hamiltonian (K^1 has a Hamilton path) and trivially satisfies the above condition. We may thus now assume that $n \geq 2$.

Let $a_1, \ldots, a_n$ be a sequence as in the statement. Denote the sequence $a_1 + 1, \ldots, a_n + 1, n$ by $a'_1, \ldots, a'_{n+1}$. Observe that $n+1 \geq 3$ and that

- $i \leq \frac{n}{2}$ is equivalent to $i < \frac{n+1}{2}$;
- $a_i < i$ is equivalent to $a'_i \leq i$;
- $a_{n+1-i} \geq n-i$ is equivalent to $a'_{n+1-i} \geq n+1-i$.

Thus, $a_1, \ldots, a_n$ satisfies the degree condition of Corollary 5.7 if and only if $a'_1, \ldots, a'_{n+1}$ satisfies the corresponding condition of Theorem 5.6, which is in turn equivalent to $a'_1, \ldots, a'_{n+1}$ being hamiltonian.

Suppose that $a_1, \ldots, a_n$ satisfies the degree condition and let G be a graph on n vertices with pointwise larger degree sequence. Define the graph G' by adding to G a vertex v that is adjacent to all other vertices. Then the degree sequence of G' is pointwise larger than $a'_1, \ldots, a'_{n+1}$, which means that G' has a Hamilton cycle, because $a'_1, \ldots, a'_{n+1}$ is hamiltonian. Deleting v from the Hamilton cycle in G' leaves a Hamilton path in G. Thus, $a_1, \ldots, a_n$ is path-hamiltonian.

Vice versa, suppose that $a_1, \ldots, a_n$ does not satisfy the degree condition. Pick a non-hamiltonian graph G' on $n+1$ vertices with degree sequence pointwise larger than $a'_1, \ldots, a'_{n+1}$ and delete a vertex v of degree $a'_{n+1} = n$ to obtain a graph G on n vertices whose degree sequence is pointwise larger than $a_1, \ldots, a_n$. If G had a Hamilton path P, from u to w say, then $uPwvu$ were a Hamilton cycle in G', a contradiction. In particular, $a_1, \ldots, a_n$ is not path-hamiltonian. □

5.2.3 A necessary condition for Hamilton cycles

Every hamiltonian graph has to be connected, even 2-connected, because its Hamilton cycle is a 2-connected spanning subgraph. In the proof of Chvátal's theorem, we saw a slightly more general necessary condition: Whenever we delete i vertices, the remaining graph is not allowed to have more than i components. This inspires the following concept.

Definition (Toughness)**.** For a real number $t > 0$, we say that a graph G is *t-tough* if, for every set $S \subseteq V(G)$ that separates G, the graph $G - S$ has at most $\frac{|S|}{t}$ components.

As we mentioned above, every hamiltonian graph is 1-tough, but the converse does not hold: Not every 1-tough graph is hamiltonian (exercise). This raises the question whether t-toughness for some fixed value of t implies hamiltonicity.

Toughness Conjecture (Chvátal 1973)**.** There exists $t > 0$ such that every t-tough graph with at least three vertices is hamiltonian.

Since 1-toughness is not enough, the value t from the Toughness Conjecture would have to be larger than 1. In fact, the Toughness Conjecture has long been expected to be true for $t = 2$, which was disproved in 2000 by Bauer, Broersma, and Veldman. Their construction is based on the graph L depicted in Figure 5.7.

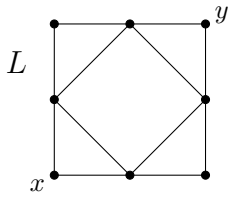

Figure 5.7: The graph L used to construct a non-hamiltonian 2-tough graph.

For every positive integer k, denote by $G_{k,2k+1}$ the graph obtained by the following construction. Take the disjoint union of a complete graph K^k and $2k+1$ copies $L_1, \ldots, L_{2k+1}$ of L, with x_i, y_i being the copies of x, y in L_i. Then connect the vertex set $\{x_1, \ldots, x_{2k+1}, y_1, \ldots, y_{2k+1}\}$ by a clique and add all edges between the copy of K^k and the copies of L.

Bauer, Broersma, and Veldman prove that $G_{k,2k+1}$ is $\frac{9k+4}{4k+3}$-tough, but does not contain a Hamilton cycle. In particular, the graph $G_{2,5}$ is still the smallest known non-hamiltonian 2-tough graph, with 42 vertices. Furthermore, by letting k tend to infinity, we see that the Toughness Conjecture can only by true for $t \geq \frac{9}{4}$.

Chapter 6

Extremal graph theory

The general theme of extremal graph theory can be summarised in the following way: If we are looking for a specific *local* substructure, e.g. a fixed subgraph or minor, how large does some *global* parameter of a graph G on n vertices, e.g. $d(G)$, $\kappa(G)$, or $\chi(G)$, have to be in order to enforce this substructure in G? Vice versa, which graphs on n vertices have the largest value of the global parameter *without* containing the substructure?

6.1 Subgraphs

Definition (Extremal graphs)**.** Let H be a fixed graph and n be a positive integer. A graph G on n vertices is called *extremal* for n and H if $H \not\subseteq G$ and $\|G\|$ is largest possible under this condition. In this case, we write $\mathrm{ex}(n, H) := \|G\|$.

Clearly, every extremal graph for n and H is in particular edge-maximal without H as a subgraph. The converse is false. For instance, a cycle on five vertices is edge-maximal without a triangle, but not extremal: The complete bipartite graph $K_{2,3}$ has more edges and is also triangle-free.

For general n, reasonable candidates for extremal triangle-free graphs are complete bipartite graphs. If we want to maximise the number of edges, then we should additionally have sides of (almost) same size, i.e. $K_{\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil}$. More generally, if we consider graphs without a K^{r+1}, the following graphs are reasonable candidates for extremal graphs.

Definition (Complete r-partite graphs, Turán graphs)**.** Given $n, r \in \mathbb{N}$ with $r \geq 1$, let $V_1, \dots, V_r$ be disjoint sets. A graph G on vertex set $V_1 \cup \dots \cup V_r$ is called *r-partite* with *sides* $V_1, \dots, V_r$ if each V_i is independent in G. (Observe that this is equivalent to being r-colourable with colour classes $V_1, \dots, V_r$.) If moreover all vertices in distinct V_i's are adjacent, we call G *complete r-partite*. A complete r-partite graph in which all sides have size s is denoted by K_s^r. We call a graph *complete multipartite* if it is complete r-partite for some r.

Suppose that $V_1, \dots, V_r$ are disjoint with

$$|V_1| + \dots + |V_r| = n \qquad \text{and} \qquad |V_1| \leq \dots \leq |V_r| \leq |V_1| + 1.$$

Then the complete r-partite graph with sides $V_1, \dots, V_r$ is called *Turán graph* and is denoted by $T^r(n)$.

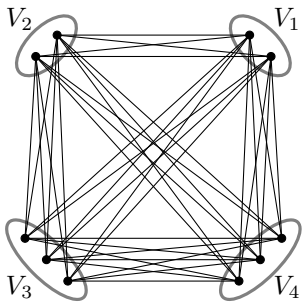

Figure 6.1: The Turán graph $T^4(10)$.

Note that $T^r(n)$ is also defined for $n < r$, in which case the sets $V_1, \dots, V_{r-n}$ will be empty and $T^r(n) = K^n$.

Theorem 6.1 (Turán 1941)**.** *For all $n, r \in \mathbb{N}$ with $r \geq 2$ and $n \geq 1$, the Turán graph $T^{r-1}(n)$ is the* unique *extremal graph for n and K^r.*

Proof. First observe that $K^r \not\subseteq T^{r-1}(n)$, because $T^{r-1}(n)$ is $(r-1)$-colourable, but K^r is not. Let G be extremal for n and K^r. We aim to prove that $G = T^{r-1}(n)$, which will imply the theorem.

Let us first prove that G is complete multipartite, which is the case if and only if "not being adjacent" is a transitive relation on $V(G)$. Suppose, for contradiction, that there are vertices $x, y, z \in V(G)$ with $xy, xz \notin E(G)$, but $yz \in E(G)$. If $d(x) < d(y)$, then the graph G' obtained from $G - x$ by adding a vertex y' with the same neighbourhood as y has n vertices and

$$\|G\| - d(x) + d(y) > \|G\|$$

edges and thus contains a K^r, because G is extremal for n and K^r. But this K^r cannot contain *both* y and y', because these vertices are not adjacent. Hence the K^r in G' gives rise to a K^r in G, a contradiction.

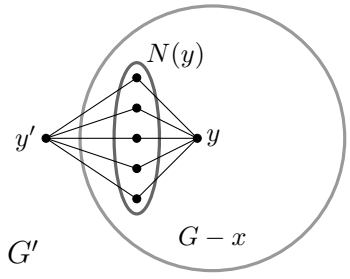

Figure 6.2: Constructing the graph G' if $d(x) < d(y)$.

Thus, $d(x) \geq d(y)$ and analogously also $d(x) \geq d(z)$. Now the graph G'' obtained from $G - y - z$ by adding two vertices x', x'' with the same neighbourhoods as x has n vertices and

$$\|G\| - (d(y) + d(z) - 1) + 2d(x) > \|G\|$$

edges (observe that the edge yz that we deleted is counted both in $d(y)$ and in $d(z)$) and thus contains a K^r, because G is extremal. Again, this gives rise to a K^r in G, a contradiction.

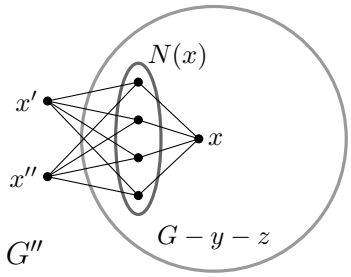

Figure 6.3: Constructing the graph G'' if $d(x) \geq d(y), d(z)$.

This means that G is complete k-partite for some k. Since $K^r \not\subseteq G$, we have $k < r$. Suppose first that $k \leq r-2$. If any side of G contains (at least) two vertices, we could add an edge between them without creating a K^r, a contradiction. Thus, $n = k \leq r-2$ and $G = K^n = T^{r-1}(n)$.

It remains to consider the case $k = r-1$. Denote the sides of G by $V_1, \ldots, V_{r-1}$, ordered by increasing size. If $|V_{r-1}| \geq |V_1| + 2$, then moving one vertex from V_{r-1} to V_1 would increase the number of edges while keeping the graph $(r-1)$-partite, and thus in particular without a K^r. This would contradict G being extremal for n and K^r. Thus, $G = T^{r-1}(n)$ as desired. □

Remark. *Turán's theorem in particular implies that*

$$\begin{aligned} \operatorname{ex}(n, K^r) = \left\|T^{r-1}(n)\right\| &= \frac{1}{2}\sum_{i=1}^{r-1} |V_i|\,(n - |V_i|) = \frac{1}{2}\left(n^2 - \sum_{i=1}^{r-1} |V_i|^2\right) \\ &= \frac{1}{2}\left(n^2 - (r-1)\left(\frac{n}{r-1}\right)^2 + O(n)\right) = \frac{1}{2}n^2\frac{r-2}{r-1} + O(n) \end{aligned} \tag{6.1}$$

with $\operatorname{ex}(n, K^r) = \frac{1}{2}n^2\frac{r-2}{r-1}$ *if and only if* n *is divisible by* $r-1$.

In other words, we know that a little bit more than $\frac{1}{2}n^2\frac{r-2}{r-1}$ edges in a graph on n vertices suffice to force K^r as a subgraph. The following result, which we state without proof, tells us that just εn^2 additional edges, no matter how small ε, guarantee the existence of K^r_s, for any s.

Theorem 6.2 (Erdős, Stone 1946)**.** *For all* $r, s \in \mathbb{N}$ *and* $\varepsilon > 0$ *with* $r \geq 2$ *and* $s \geq 1$*, there exists* $n_0 \in \mathbb{N}$ *such that every graph* G *on* $n \geq n_0$ *vertices with*

$$\|G\| \geq \frac{1}{2}n^2\frac{r-2}{r-1} + \varepsilon n^2$$

contains K^r_s *as a subgraph.*

As an immediate corollary of Theorem 6.2, we get that

$$\lim_{n\to\infty} \frac{\operatorname{ex}(n, K^r_s)}{\binom{n}{2}} = \frac{r-2}{r-1}.$$

In fact, Theorem 6.2 can be used to determine the asymptotic value of $\mathrm{ex}(n, H)$ for almost all graphs H.

Theorem 6.3 (Erdős, Simonovits 1966)**.** *For every graph H with at least one edge,*

$$\lim_{n\to\infty} \frac{\mathrm{ex}(n, H)}{\binom{n}{2}} = \frac{\chi(H) - 2}{\chi(H) - 1}.$$

Proof. Write $r := \chi(H)$ and let $n \in \mathbb{N}$. Then $H \not\subseteq T^{r-1}(n)$, because $T^{r-1}(n)$ is $(r-1)$-colourable while H is not. On the other hand, we can pick s large enough so that $H \subseteq K_s^r$. By (6.1) and Theorem 6.2, we have

$$\frac{r-2}{r-1} = \lim_{n\to\infty} \frac{\left\|T^{r-1}(n)\right\|}{\binom{n}{2}} \leq \liminf_{n\to\infty} \frac{\mathrm{ex}(n, H)}{\binom{n}{2}} \leq \limsup_{n\to\infty} \frac{\mathrm{ex}(n, H)}{\binom{n}{2}} \leq \frac{r-2}{r-1} + 2\varepsilon$$

for every $\varepsilon > 0$. Now $\varepsilon \to 0$ proves the theorem. □

If H is an edgeless graph, then clearly $\mathrm{ex}(n, H) = 0$ for $n \geq |H|$. For bipartite graphs H, Theorem 6.3 tells us that $\mathrm{ex}(n, H) = o(n^2)$. It turns out that for some bipartite graphs, we can make this more precise.

Theorem 6.4. *For every integer $r \geq 2$, there exist constants $c_1, c_2 > 0$ (depending on r) such that*

$$c_1 n^{2-\frac{2}{r+1}} \leq \mathrm{ex}(n, K_{r,r}) \leq c_2 n^{2-\frac{1}{r}}$$

for every $n \in \mathbb{N}$.

Observe that Theorem 6.4 does *not* state the asymptotic order of $\mathrm{ex}(n, K_{r,r})$, only an upper and a lower bound. The lower bound is proved using probabilistic methods. The upper bound can be proved directly.

Proof of the upper bound. Let G be an extremal graph for n and $K_{r,r}$. The basic idea behind this proof will be to double count the number of subgraphs K of G are isomorphic to $K_{1,r}$.

That is, we are counting the number of pairs

$$\{(v, X) \colon x \in V(G), X \subseteq N(v) \text{ and } |X| = r\}.$$

For each vertex v there are exactly $\binom{d(v)}{r}$ many $r-$sets X with $X \subseteq N(a)$. Hence the total number of such pairs is $\sum_{v\in V(G)} \binom{d(v)}{r}$.

On the other hand, for each fixed r-set X, since G contains no $K_{r,r}$, there are at most $r-1$ vertices v with $X \subseteq N(v)$. Hence we see that

$$\sum_{v\in V(G)} \binom{d(v)}{r} \leq (r-1)\binom{n}{r} \leq rn^r. \tag{6.2}$$

Hence, it would be useful to bound the left hand side of the above equation below as a function of the number of edges m of G. A first useful step will be to note that

$$\sum_{v\in V(G)} \binom{d(v)}{r} \geq \sum_{v\in V(G)} \left(\frac{d(v)}{r}\right)^r - 1,$$

where we've used that $\binom{d(v)}{r} \geq \left(\frac{d(v)}{r}\right)^r$ if $d(v) \geq r$ and $\frac{d(v)}{r} < 1$ if $d(v) \leq r$.

One way to bound this quantity is then to use the *power mean inequality*, which is a generalisation of the AM-GM inequality, which says that for any set $\{x_1, x_2, \ldots, x_t\}$ of positive real numbers and any real numbers $i < j$

$$\left(\sum_{k=1}^{t} \frac{x_k^i}{t}\right)^{\frac{1}{i}} \leq \left(\sum_{k=1}^{t} \frac{x_k^j}{t}\right)^{\frac{1}{j}}$$

Applying this to the number $\{d(v)\colon v \in V(G)\}$, so that $t = n$, with $i = 1$ and $j = r$ we see that

$$\begin{aligned}
\sum_{v \in V(G)} \left(\frac{d(v)}{r}\right)^r &= \frac{n}{r^r} \sum_{v \in V(G)} \frac{d(v)^r}{n} \\
&\geq \frac{n}{r^r} \left(\sum_{v \in V(G)} \frac{d(v)}{n}\right)^r \\
&\geq \frac{n}{r^r} \left(\frac{2m}{n}\right)^r \\
&\geq \frac{n^{1-r}(2m)^r}{r^r}
\end{aligned}$$

Hence, by (6.2) we see that

$$rn^r \geq \frac{n^{1-r}(2m)^r}{r^r}$$

and re-arranging leads to

$$||G|| = m \leq \frac{r^{1+r}}{2} n^{2-\frac{1}{r}},$$

and so we can take $c_2 = \frac{r^{1+r}}{2}$.

□

Remark. *For a tree T (with at least two edges), $\mathrm{ex}(n, T)$ is linear in n (exercise). A conjecture by Erdős and Sós states that $\frac{1}{2}(\|T\| - 1)n$ should be an upper bound for $\mathrm{ex}(n, T)$.*

6.2 Minors and topological minors

In contrast to subgraphs, minors and topological minors can be enforced by a linear number of edges. For this reason, it is convenient to state the corresponding results in terms of the average degree instead of the number of edges.

Proposition 6.5. *Let r be a positive integer. Every graph with average degree at least $2^{\binom{r}{2}}$ contains a TK^r.*

Proof. For $r \leq 2$, the statement is trivial. We may thus assume that $r \geq 3$. We now prove by induction on $k = r, \ldots, \binom{r}{2}$ that every graph with average degree at least 2^k contains a TX for some graph X with r vertices and k edges.

For $k = r$, let G be a graph with average degree $d(G) \geq 2^r$. Then G contains a subgraph G' with $\delta(G') > 2^{r-1}$ (recursively deleting vertices of degree at most 2^{r-1} does not decrease the average degree). A longest path P in G', with first vertex v say, contains all $d_{G'}(v) \geq 2^{r-1} + 1$ neighbours of v. If w is the last neighbour of v on P, then $vPwv$ is a cycle of length at least $2^{r-1} + 2$, which is larger than r. Thus, the claim follows for X being a cycle of length r.

For the induction step, let $r < k \leq \binom{r}{2}$ and suppose that the claim is true for all smaller values of k. Let G with $d(G) \geq 2^k$ be given. Without loss of generality, we may assume that G is connected, because every non-connected graph has a component with average degree at least the average degree of the full graph. Let $U \subseteq V(G)$ be a largest set with

- U is connected in G and
- $d(G/U) \geq 2^k$.

Such a set exists, because every singleton satisfies these properties.

We have $N(U) \neq \emptyset$, since G is connected and $U \neq V(G)$; set $H := G[N(U)]$. We claim that $\delta(H) \geq 2^{k-1}$. For if v were a vertex in H of degree $d_H(v) < 2^{k-1}$, then setting $U' := U \cup \{v\}$ would result in

$$\|G/U'\| = \|G/U\| - (d_H(v) + 1) \geq 2^{k-1}(|G/U| - 1) = 2^{k-1}\,|G/U'|\,.$$

(Compared with G/U, we lose precisely the edges from v to other vertices in H, plus the edge to the contracted vertex U.) Now U' would be connected and $d(G/U') \geq 2^k$, contradicting the maximality of U. Thus, the minimum degree—and hence also the average degree—of H is at least 2^{k-1}.

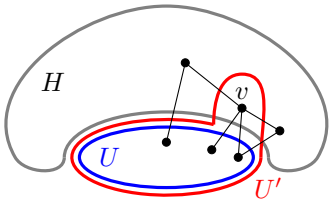

Figure 6.4: If we add a vertex v to U, then any edges from v to other vertices in H are lost in G/U' compared to G/U.

By the induction hypothesis, H contains a TY, where Y is a graph with r vertices and $k - 1$ edges. Let x, y be two non-adjacent vertices of Y. The corresponding vertices v_x, v_y in TY have neighbours in U and thus can be connected in G via a path through U. Adding this path to the TY yields a TX with $X = Y + xy$, concluding the proof. □

As a surprising corollary of Proposition 6.5, we deduce a result about connectivity of graphs. In fact, we shall prove a much stronger property.

Definition (Linkability)**.** Let $k \in \mathbb{N}$. A graph G with $|G| \geq 2k$ is called *k-linked* if for every choice of distinct vertices $s_1, \dots, s_k, t_1, \dots, t_k$, there exist disjoint paths $P_1, \dots, P_k$ in G such that each P_i is an s_i–t_i path.

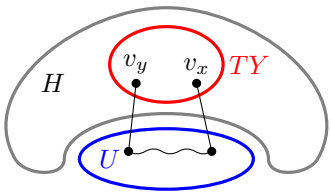

Figure 6.5: Two branch vertices can be connected via a path through U.

Observe that if we set $S := \{s_1, \dots, s_k\}$ and $T := \{t_1, \dots, t_k\}$, then k-connectedness of G would, by Menger's theorem, give us k disjoint S–T paths. However, we would not be able to control which vertices are connected by these paths. Thus, being k-linked is a stronger property than being k-connected.

Proposition 6.6. *Let $k \in \mathbb{N}$. Every $2^{\binom{3k}{2}}$-connected graph G is k-linked.*

Proof. Let k and G be fixed as in the statement. The theorem is trivial for $k = 0$, so assume that k is positive. By Proposition 6.5 and the fact that $d(G) \geq \delta(G) \geq \kappa(G)$, there is a subgraph $K = TK^{3k}$ of G. Denote by U the set of branch vertices of K.

Suppose that distinct vertices $s_1, \dots, s_k, t_1, \dots, t_k$ in G are given. Since $\kappa(G) \geq 2^{\binom{3k}{2}} \geq 2k$, Menger's theorem gives us a set $\mathcal{P}$ of $2k$ disjoint paths $Q_1, \dots, Q_k, R_1, \dots, R_k$ such that each Q_i is an s_i–U path and R_i is a t_i–U path. Suppose that we have chosen $\mathcal{P}$ so that the number of edges that these paths have outside K is as small as possible.

Out of the $3k$ vertices in U, precisely $2k$ are end vertices of paths in $\mathcal{P}$; let $u_1, \dots, u_k$ be the remaining vertices in U. Let L_i be the subdivided edge of K^{3k} in K from u_i to the end vertex of Q_i in U (in this direction). Denote by v_i the first vertex on L_i that lies on any path $Q \in \mathcal{P}$. By the choice of $\mathcal{P}$, replacing Q by $Qv_iL_iu_i$ does not decrease the number of edges these paths have outside K. This means that $v_iQ \subseteq K$ and thus $Q = Q_i$. Analogously, the first vertex w_i on the subdivided edge M_i of K^{3k} in K from u_i to the end vertex of R_i in U that lies on any path in $\mathcal{P}$ satisfies $w_i \in R_i$.

Now

$$P_i := s_iQ_iv_iL_iu_iM_iw_iR_it_i$$

is an s_i–t_i path and all P_i's are pairwise disjoint, proving that G is k-linked. □

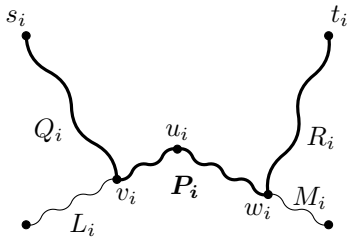

Figure 6.6: Rerouting Q_i and R_i via u_i to obtain an s_i–t_i path P_i.

The bounds in Propositions 6.5 and 6.6 are in fact far larger than necessary. We now state (without proof) the best bounds known up to date.

Theorem 6.7 (Komlós, Szemerédi 1996; Bollobás, Thomason 1998)**.** *There exists a constant $c > 0$ such that, for every $r \in \mathbb{N}$, every graph with average degree at least cr^2 contains a TK^r.*

The quadratic degree is necessary to enforce a TK^r: there exist graphs with average degree $\frac{1}{8}r^2$ and no TK^r.

Theorem 6.8 (Thomas, Wollan 2005)**.** *Let $k \in \mathbb{N}$. Every $2k$-connected graph G with $d(G) \geq 16k$ is k-linked. In particular, every $16k$-connected graph is k-linked.*

Let us now consider minors instead of topological minors. Since every topological minor is also a minor, quadratic average degree is enough to guarantee an MK^r. But in fact, a much smaller degree is already sufficient.

Theorem 6.9 (Kostochka 1982)**.** *There exists a constant $c > 0$ such that, for every integer $r \geq 2$, every graph with average degree at least $cr\sqrt{\log_2 r}$ contains an MK^r. This bound is best possible as a function in r (up to the choice of c).*

6.3 Hadwiger's Conjecture

If $\chi(G) \geq cr\sqrt{\log_2 r} + 1$, then Corollary 4.11(ii) tells us that G has a subgraph of minimum degree (and thus also average degree) at least $cr\sqrt{\log_2 r}$, which in turn contains an MK^r by Theorem 6.9. It is conjectured that in fact a much smaller chromatic number already suffices to guarantee an MK^r.

Hadwiger's Conjecture (Hadwiger 1943)**.** Let r be a positive integer. Then every graph with chromatic number at least r contains an MK^r.

Remark. *If $r_1 > r_2$, then Hadwiger's Conjecture for $r = r_1$ implies the conjecture for $r = r_2$ (exercise).*

Let us now look at specific values of r for which the conjecture is known to be true. The proofs will gradually become more difficult.

$r \leq 3$. Hadwiger's Conjecture is trivial for $r = 1$ (every graph with non-zero chromatic number is non-empty; thus each vertex is a K^1) and for $r = 2$ (every graph with chromatic number at least two has an edge, i.e. a K^2). For $r = 3$, observe that every cycle in a graph forms an MK^3. Thus, the only graphs without MK^3 are forests, which are 2-colourable.

$r = 4$. For this case of Hadwiger's Conjecture, we use a structural result about graphs that are edge-maximal without containing an MK^4. Before we can phrase this result, we need the following terminology.

Definition (Pasting graphs)**.** Let G_1, G_2 be induced subgraphs of a graph G and write $H := G_1 \cap G_2$. If $G = G_1 \cup G_2$, then we say that G is obtained by *pasting G_1 and G_2 along H*.

Proposition 6.10. *A graph on at least three vertices is edge-maximal without an MK^4 if and only if it can be constructed from triangles by recursively pasting along edges.*

Proof. Like in the proof of Proposition 3.8, where we (among other things) proved that every $MK_{3,3}$ contains a $TK_{3,3}$, we see that containing an MK^4 is equivalent to containing a TK^4. We may thus replace "edge-maximal without MK^4" by "edge-maximal without TK^4".

Let us first prove the following auxiliary result.

> If a graph G is obtained by pasting graphs G_1, G_2 along an edge, and if $TK^4 \not\subseteq G_1$ and $TK^4 \not\subseteq G_2$, then also $TK^4 \not\subseteq G$. $(*)$

To prove $(*)$, let G be obtained by pasting G_1, G_2 along an edge xy and suppose that G contains a TK^4. One of G_1, G_2, without loss of generality G_1, contains all four branch vertices, because branch vertices $b_1 \in V(G_1 - G_2)$ and $b_2 \in V(G_2 - G_1)$ would be separated by $\{x, y\}$ and thus be connected by at most two internally disjoint paths, contradicting the fact that there are three internally disjoint b_1–b_2 paths in the TK^4. At most one of the subdivided edges of K^4 can pass through $G_2 - G_1$; replacing such a segment by the edge xy yields a TK^4 in G_1. This proves $(*)$.

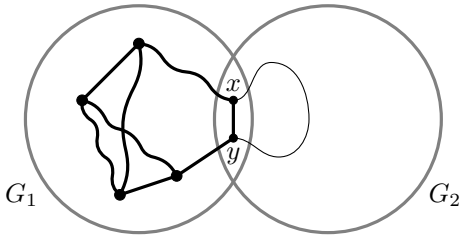

Figure 6.7: Generating a TK^4 in G_1 from a TK^4 in G.

We now prove the proposition by induction on $|G|$. The induction base $|G| = 3$ is trivial, because triangles are both edge-maximal without TK^4 and constructible. Now let $|G| \geq 4$.

Assume first that G is constructible; i.e. it is obtained by pasting two constructible graphs G_1, G_2 along an edge xy. Then $(*)$ implies that G does not contain a TK^4. To prove that G is *edge-maximal* without TK^4, let $v_1, v_2 \in V(G)$ be non-adjacent vertices. If v_1, v_2 lie in the same G_i, then $G + v_1v_2$ contains a TK^4, because G_1 and G_2 are edge-maximal without TK^4. We may thus assume that $v_1 \in V(G_1 - G_2)$ and $v_2 \in V(G_2 - G_1)$. An easy induction argument (using Lemma 1.10) shows that G_1, G_2 are 2-connected. By the fan version of Menger's theorem, we find a v_1–$\{x, y\}$ fan in G_1 and a v_2–$\{x, y\}$ fan in G_2, consisting of two paths each. The union of these paths, together with the edges v_1v_2 and xy, forms a TK^4 in $G + v_1v_2$, with branch vertices v_1, v_2, x, y. Thus, G is edge-maximal without TK^4.

Vice versa, suppose that G is edge-maximal without TK^4, then it is in particular connected. Since $|G| \geq 4$, we also know that G is not complete; let S be a smallest separator and let u, v be vertices in $N(S)$ separated by S. Let U be the vertex set of the component of $G - S$ that contains u and set $G_1 := G[U \cup S]$ and $G_2 := G - U$. We aim to show that $G_1 \cap G_2 = G[S]$ is an edge and that both G_1 and G_2 are constructible.

If $|S| \geq 3$, then u, v are connected by internally disjoint paths P_1, P_2, P_3 by Menger's theorem. The set $\{u, v\}$ is not a separator, thus we can pick a shortest

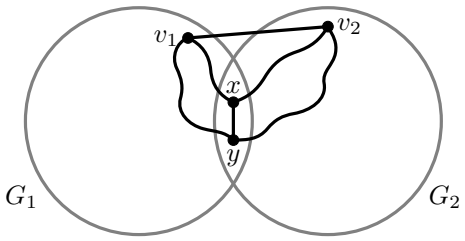

Figure 6.8: Finding a TK^4 in $G + v_1v_2$.

path Q between vertices w_i, w_j of distinct P_i, P_j that avoids $\{u, v\}$. This yields a TK^4 with branch vertices u, v, w_i, w_j, a contradiction.

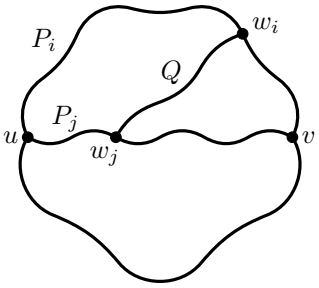

Figure 6.9: In the case $|S| \geq 3$, we find a TK^4 in G by applying Menger's theorem.

If S is just a single vertex w, then $G + uv$ is obtained by pasting G_1 and $G_2 \cup vuw$ along an edge. By assumption, G_1 does not contain a TK^4. If $G_2 \cup vuw$ contained a TK^4, then u would not be a branch vertex (since it has degree two) and any subdivided edge passing through u could be shortcut via vw to yield a TK^4 in G_2, a contradiction. Thus, $(*)$ implies that $G + uv$ does not contain a TK^4, a contradiction.

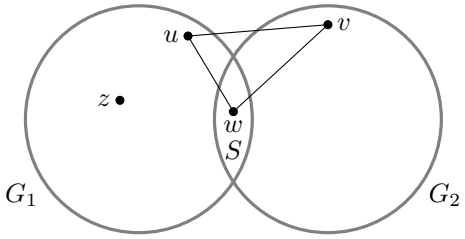

Figure 6.10: In the case $|S| = 1$, there is no TK^4 in $G + uv$, because neither G_1 nor $G_2 + vuw$ contains a TK^4.

This means that $S = \{s, t\}$. If s, t were not adjacent, then $G + st$ would contain a TK^4 and would be obtained by pasting $G_1 + st$ and $G_2 + st$ along an edge. But then by $(*)$, one of the two parts, say $G_1 + st$, would contain a TK^4 as well. This TK^4 needs to use the edge st; the subdivided edge that goes through st could then be rerouted via G_2 to give rise to a TK^4 in G, a

contradiction. Thus, st is an edge. This means that G is obtained by pasting G_1 and G_2 along an edge.

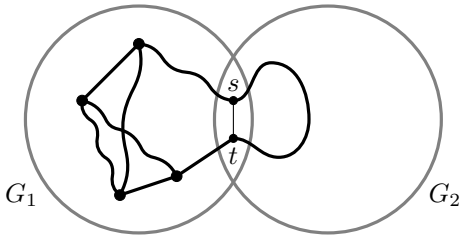

Figure 6.11: Generating a TK^4 in G from a TK^4 in $G + st$.

To show that G_1 is constructible, first note that G_1 contains no TK^4, because G does not. If $u, v \in V(G_1)$ are not adjacent, then $G + uv$ contains a TK^4 and is obtained by pasting $G_1 + uv$ and G_2 along an edge. Since $TK^4 \not\subset G_2$, we deduce that $G_1 + uv$ contains a TK^4 by $(*)$. Thus, G_1 (and analogously also G_2) is edge-maximal without a TK^4. By induction, both graphs are constructible and thus so is G, as desired. □

Corollary 6.11. *Hadwiger's Conjecture holds for $r = 4$.*

Proof. Triangles are 3-colourable and pasting two 3-colourable graphs along an edge results in a 3-colourable graph. Therefore, Proposition 6.10 implies that all edge-maximal graphs (and thus all graphs) without an MK^4 are 3-colourable. In other words, every graph with chromatic number at least four has to contain an MK^4. □

$r = 5$. Edge-maximal graphs without an MK^5 can be constructed in a fashion similar to Proposition 6.10.

Theorem 6.12 (Wagner 1937)**.** *A graph G with $|G| \geq 3$ is edge-maximal without an MK^5 if and only if it can be constructed from the "Wagner graph"*

$$W := \big(\{v_1, \dots, v_8\}, \{v_i v_j : i - j \in \{\pm 1, 4\} \bmod 8\}\big)$$

and plane triangulations by recursively pasting along triangles or edges.

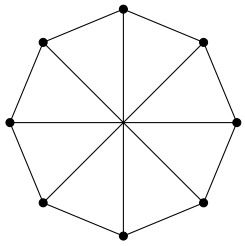

Figure 6.12: The Wagner graph W.

The proof of Theorem 6.12 is considerably longer than that of Proposition 6.10, but is using similar ideas. We omit the proof in this course.

Corollary 6.13. *Hadwiger's Conjecture holds for $r = 5$.*

Proof. The Wagner graph is 3-colourable, while every plane triangulation is 4-colourable by the Four Colour Theorem. Thus, Theorem 6.12 implies that every graph without an MK^5 is 4-colourable. Vice versa, this means that every graph with chromatic number at least five contains an MK^5. □

Remark. *We saw in the proof of Corollary 6.13 that the Four Colour Theorem implies Hadwiger's Conjecture for $r = 5$. In fact, the reverse implication holds as well, with a much easier argument: Hadwiger's Conjecture for $r = 5$ states that every graph with chromatic number at least five contains an MK^5 and thus is not planar, i.e. every planar graph is 4-colourable.*

Any proof of Hadwiger's Conjecture for $r \geq 5$ (recall Hadwiger's Conjecture for any fixed value $r \geq 5$ would imply the case $r = 5$, see the remark on page 104) that does not *use the Four Colour Theorem would thus reprove the Four Colour Theorem. To this date, no such proof is known.*

***r* = 6.** This is the largest value for which Hadwiger's Conjecture has been proved.

Theorem 6.14 (Robertson, Seymour, Thomas 1993)**.** *Hadwiger's Conjecture holds for $r = 6$.*

The proof of this case is again much harder than for any smaller r. The basic proof idea is as follows. Suppose that the conjecture is false and let G be a smallest counterexample. Robertson, Seymour, and Thomas then need several case distinctions and about 80 pages to prove that G has to contain a vertex v such that $G - v$ is planar. By the Four Colour Theorem, this contradicts the assumption that $\chi(G) \geq 6$.

Remark. *Hadwiger's Conjecture is open for every $r \geq 7$.*