Lecture notes: **"Graph Theory 2"**

December 7, 2018

Anders Nedergaard Jensen

Preface

In this course we study algorithms, polytopes and matroids related to graphs. These notes are almost complete, but some of the examples from the lectures have not been typed up, and some background information, such as proper definitions are sometimes lacking. For proper definitions we refer to: Bondy and Murty: "Graph Theory". Besides these minor problems, the notes represent the content of the class very well.

Contents

Introduction, summary and conventions

If we regard a simple graph as just a subset $E \subseteq V \times V$, we a priori have little mathematical structure to play with. As a result many proofs are ad hoc and graph theory becomes a difficult field to master. In this class we present one solution to this problem, namely we will study graphs in terms of *polytopes* and *matroids*. This philosophy fits well with the *combinatorial optimisation* approach to graph theory. We will soon realise that various algorithms for optimisation problems involving graphs are closely related to general standard techniques in *mathematical programming* such as separation, linear programming (LP), LP-duality, ellipsoid methods, relaxations and integer linear programming (ILP). In the following paragraphs we give an overview of some main results of this course. The overview will make most sense at the end of the class.

An important aspect of algorithms is that of producing *certificates* of correctness. Starting out with Egerváry's bipartite maximum matching algorithm (Algorithm 1.3.9), we see that indeed the correctness of its result is certified by a minimum covering. For Edmonds' blossom algorithm (Algorithm 1.4.8) for the non-bipartite case such a certificate is a so called *barrier*. Egerváry's and Edmonds' algorithms both work by finding *augmenting paths*. Non-existence of augmenting paths implies optimality (Berge's Theorem 1.2.3).

While the weighted version of the bipartite matching problem can be solved as a linear program with a reasonable number of inequalities, we also present the Hungarian algorithm (Algorithm 1.5.3) which runs in polynomial time. For the non-bipartite weighted case, the situation is more complicated as the required number of inequalities for describing the *perfect matching polytope* is exponential (Theorem 1.7.6). Therefore we rely on a quick separation algorithm (Algorithm 1.7.16) by Padberg and Rao and the ellipsoid method (Section 1.7.4). The ellipsoid method is more of theoretical interest than it is practical. A practical method is out of the scope of this course. Finally, maximal weight matching is applied to the Chinese postman problem (Algorithm 1.7.2).

In connection to the matching problems, we need algorithms for finding distances in graphs. While Dijkstra's algorithm (Algorithm 1.6.1) is assumed to be known, these notes contain several other algorithms (Algorithm 1.6.4 (Warshall), Algorithm 1.6.5 (Bellman-Ford-Moore) and Algorithm 1.6.12 (Floyd)), which are either faster in certain situations or deal with negative weights. These algorthims were presented in the "Graph Theory 1" class. Therefore, in class, we will try only to get on overview of the algorithms when needed, for example when they are used in Algorithm 1.5.3 and Algorithm 1.7.2. In particular, the exam topics this year will not be the same as for the 2016 exam, see Appendix B.

The second half of the course is concerned with the theory of matroids. As we are interested in optimisation, we quickly turn the *independent set* characterisation (Definition 2.4.2) into one in terms of matroid polytopes (Definition 2.7.2 and Theorem 2.7.4) and a characterisation in terms of the *greedy algorithm* (Algorithm 2.8.2 and Theorem 2.8.4). Assuming positive objective function, we can think of the greedy algorithm as solving an LP problem over a matroid polytope.

The question of whether a graph has a Hamiltonian path can be phrased as

a question about the largest size of a common independent set of three matroids (Algorithm 2.10.4). If we think of the matroids as being represented by oracles, we conclude that matroid intersection for three matroids is very difficult (the Hamiltonian cycle problem is one of Karp's original 21 NP-hard problems). Surprisingly, the situation is much simpler for two matroids. We do not present an algorithm but just see that Hall's Theorem 1.2.20 has generalisations to matroids (Theorem 2.9.6) and refer to [10] for an algorithm. As an example we study the optimum branching problem (Section 3.1) and give an effective algorithm. We then notice that this can be phrased as an intersection problem of two matroids (Exercise 3.1.22).

Finally, we discuss various graph colouring themes: Brooks' Theorem 4.1.5, chromatic polynomials (Section 4.2) and planar graph colourings (Section 4.3).

It comes as no surprise that the combinatorial graph problems have applications in *operations research*. We give some examples of this through the course (although some situations seem somewhat artificial): Exercise 1.1.3, Section 1.7.1, Exercise 3.1.19.

Conventions

Remark 0.0.1 Consider a set V. An *edge* is an unordered pair $e = (u, v)$ with $u, v \in V$. The elements of V are called vertices, while the vertices u and v are called the *ends* of e. If $u = v$ then e is called a *loop*. If E is a collection of edges, then $G = (V, E)$ is called a graph. We will almost always assume that E is a set and not a multiset, meaning that there can be only one copy a particular unordered pair (u, v) - in other words, most of the time we do not allow *parallel* edge i.e. edges with the same set of ends. In only a few places we make an exception and allow parallel edges (e.g. Section 1.7).

This is different from the Graph Theory 1 course, where parallel edges were allowed and we had to assign names to each edge to allow us to distinguish it from other edges with the same ends. To ease notation, often one chooses to draw graphs rather than writing down the set of vertices and edges.

Remark 0.0.2 Similarly, by a *directed* graph we mean a pair $G = (V, E)$ where E is a set of *ordered* pairs $(u, v) \in V \times V$ called *arcs*. A more typical notation is to use the letter A instead of E. When discussing graphs, it is always important to be aware of whether we are considering directed or undirected graphs.

Remark 0.0.3 We will always assume that the vertex and edge sets V and E of a graph $G = (V, E)$ are finite.

1 Matchings

In this section we study matching problems carefully. One purpose will be to convince ourselves that "studying polytopes in graph theory is OK".

1.1 Matchings

Definition 1.1.1 A matching in a graph $G = (V, E)$ is a subset $M \subseteq E$ such that M contains no loops and no two edges of M share an end.

We say that a matching M is *maximum* if $|M|$ is as large as possible. Similarly, if we have a function $w : E \to \mathbb{R}$ (called a weight function), a *maximal weight matching* in G is a matching M in G with $w(M) := \sum_{e \in M} w(e)$ as large as possible. (In contrast to these, a *maximal matching* is a matching not contained in any larger matching. This notion is only of limited interest.)

We will consider four different situations:

- Maximum matching in a bipartite graph.
- Maximum matching in a general graph.
- Maximal weight matching in a bipartite graph.
- Maximal weight matching in a general graph.

Since we only consider finite graphs, there exist algorithms for all four cases. Simply look at all possible subsets of the edges and see which ones are matchings and pick the one with largest weight or largest size, respectively. This is a terrible algorithm since the number of subsets of E is exponential in $|E|$. Our goal is to present better (polynomial time) algorithms.

Definition 1.1.2 We say that an edge e in a matching M in a graph $G = (V, E)$ *covers* a vertex $v \in V$ if e is incident to e, meaning that v is an end of e. Similarly we say that "M covers v" or "v is M-covered" if some $e \in M$ covers e. A matching M is called *perfect* if all vertices are M-covered i.e. if $2|M| = |V|$.

Exercise 1.1.3 In Saarbrücken 160 students are taking the Analysis 1 class. They need to attend TA sessions (TØ). There are 8 different time slots to choose from, say Tuesday 12-14 or Thursday 14-16 and so on. Each slot has room for 20 students. Using an online form the students choose their first priority, second priority and third priority. When a student is assigned his/her first priority we get three *points of happiness*. Similarly we get two points for a second priority, one point for a third priority and zero points if we had to choose a time that did not suit the student. Based on the data from the online form we want to find an *assignment* of students to classes maximising happiness.

1. Phrase this problem as a weighted matching problem. (Hint: You may want to consider a graph with 320 vertices.)
2. How would you phrase this matching problem as an Integer Linear Programming problem (ILP)?

3. What is a doubly stochastic matrix?

4. Will the *Linear programming* relaxation have its optimum attained at integral points? Is the feasible region of the LP a lattice polytope? (Hint: prove that the set of doubly stochastic matrices is the convex hull of 0-1 doubly stochastic matrices.)

5. For how many students would you expect this problem to be solvable with Dantzig's Simplex Method?

6. Can matching problems be used to schedule more classes simultaneously? Can linear programming problems?

Solution.

1. We model the problem as a weighted matching problem on the complete bipartite graph on 160+160 vertices. There is one vertex for each student and one vertex for each seat in each class. Weights are assigned to the $160 \cdot 160$ edges according to the students priorities. A weight of 0 is assigned to edges connecting a student to a seat in a class which the student did not have as a priority.

2. Phrased as an ILP we let $x \in \mathbb{Z}^{160 \cdot 160}$ be a vector with one entry per edge. It will represent a matching M with $x_e = 1$ if and only if $e \in M$ – and zero otherwise. We are looking for an x such that $\omega \cdot x$ is maximal subject to $Ax = (1, \ldots, 1)^t$ and $x \in \mathbb{Z}^{160 \cdot 160}$ where A is the incidence matrix of the graph.

The answers to 3 and 4 follow from Exercise 16.2.19 on page 430 in [1]. See also [6]. An alternative answer can be found in [6, Theorem 4.26]. Question 6 is quite open-ended.

1.2 When is a matching maximum?

Definition 1.2.1 Let M be a matching in a graph G. A path P in G is called an *M-alternating path* if it alternately contains an edge from M and an edge from $E(G) \setminus M$. Let x and y be two different vertices. An M-alternating xy-path[1] is called *M-augmenting* if both x and y do not appear as ends of edges in M.

A *spanning* subgraph of a graph $G = (V, E)$ is a subgraph $H = (V, F)$ where $F \subseteq G$. That is, when talking about spanning subgraphs, we are not really interested in the vertex set since that is always the same as the original graph. On the other hand a subset of the edge set uniquely determines a spanning subgraph. In particular, a matching M in $G = (V, E)$ gives rise to the spanning subgraph (V, M).

The *symmetric difference* of two sets A and B is defined as $A \Delta B := (A \cup B) \setminus (A \cap B)$. Let F and F' be subgraphs of a graph G. Then their symmetric difference $F \Delta F'$ is the (spanning) subgraph of G with edge set $E(F) \Delta E(F')$.

[1] An xy-path is a path starting at x and ending at y.

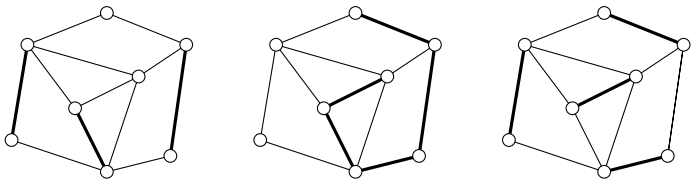

Figure 1: A matching M in a graph G. An M-augmenting path P in G. The symmetric difference of M and P.

As the following example suggests, taking symmetric difference of a matching M with an M-augmenting path P always gives a matching $M' := M\Delta P$ with $|M| + 1$ edges.

Example 1.2.2 In Figure 1 the size of a matching M increases from 3 to 4 by taking symmetric difference with an M-augmenting path P.

Theorem 1.2.3 (Berge's Theorem, [1, 16.3]) *Let M be a matching in a graph G. The matching M is a maximum matching if and only if G does not contain an M-augmenting path.*

Proof. $\Rightarrow$: If G contains an M-augmenting path P, then $P\Delta M$ is a matching in G. Moreover, $|P\Delta M| = |M| + 1$ because P contains one more edge from $E \setminus M$ than from M.

$\Leftarrow$: If M was not a maximum matching. We must show that an augmenting path exists. Let M' be a maximum matching. Then $H := M\Delta M'$ is not empty and H must contain more edges from M' than from M. Since every vertex in the edge-induced subgraphs [2] $G[M]$ and $G[M']$ has degree 1, the degrees of the vertices of $G[H]$ is 0, 1 or 2. We conclude that $G[H]$ is the union of cycles and paths. These are all M-alternating. Because H contains more edges from M' than from M, one of the components[3] of $G[H]$ is a path P starting and ending with an edge from M'. Let u be one end of P and $e' \in M'$ the incident edge. Then u is not an end of an edge $e \in M$ (If it were, then because $e \notin E(H)$ we get $e \in M'$, contradicting that M' is a matching.). Similarly the other end of P is not matched in M. We conclude that P is M-augmenting. □

1.2.1 Coverings

If an edge e covers a vertex v we also say that v covers e.

Definition 1.2.4 Let $G = (V, E)$ be a graph. A covering of G is a subset $K \subseteq V$ such that every edge of E is covered by some vertex of K.

Exercise 1.2.5 The graph G in Figure 2 has a covering consisting of the three red vertices. Find all coverings of G.

[2] See Appendix A.
[3] See Appendix A.

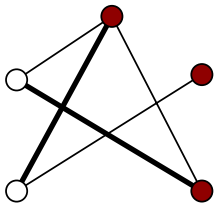

Figure 2: A graph G with a matching consisting of the two (bold) edges. The non-bold edges do not form a matching. The red vertices form a covering of G, while the white do not. The covering number happens to be $\beta(G) = 3$ and matching number $\alpha'(G) = 2$.

Definition 1.2.6 The covering number $\beta(G)$ of a graph is the minimal number of vertices in any covering of the graph G.

Example 1.2.7 The covering number of the graph in Figure 2 is $\beta(G) = 3$.

Definition 1.2.8 The matching number $\alpha'(G)$ of a graph G is the maximal number of edges in a matching of G.

Exercise 1.2.9 Show that the matching number of the graph G in Figure 2 is $\alpha'(G) = 2$ by listing all matchings in G. How many matchings are there?

Example 1.2.10 Let n be even and let G_n be the $n \times n$ grid graph ([1, page 30]) $P_n \square P_n$, but with two diagonally opposite vertices deleted. What is the matching number of G_n? (Hint: can a chess board with two opposite corners be tiled by (64-2)/2=31 domino pieces?) What is the covering number of G_n?

Lemma 1.2.11 *Let G be a graph with matching M and covering K. Then $|M| \leq |K|$.*

Proof. Each $v \in K$ can only cover a single edge of M. To cover the whole graph, X mush have at least $|M|$ vertices. □

Proposition 1.2.12 *For every graph G its matching number is at most its covering number:*

$$\alpha'(G) \leq \beta(G).$$

Proof. Because we only consider finite graphs, G must have a maximum matching M^* and a minimum covering K^*. Then by Lemma 1.2.11:

$$\alpha'(G) = |M^*| \leq |K^*| = \beta(G)$$

as desired. □

In Example 1.2.10 above, the remaining 30 white squares serve as a covering of G_8.

Exercise 1.2.13 Come up with a small graph G where $\alpha'(G) \neq \beta(G)$.

Theorem 1.2.14 *(König-Egerváry) For any bipartite graph G, we have $\alpha'(G) = \beta(G)$.*

Notice that the graph in Example 1.2.10 is bipartite.

Exercise 1.2.15 Let G be a graph. Argue that a maximum matching in G can be found by solving the following Integer Linear Programming problem:

$$\textbf{maximise } (1, \ldots, 1)^t \cdot x$$

$$\textbf{subject to } Ax \leq (1, \ldots, 1)^t \text{ and } x \in \mathbb{N}^{|E(G)|}$$

where A is the incidence matrix of G. (Notice: In Exercise 1.1.3 we knew that the matching we were looking for would match all vertices and there we had and equality $Ax = (1, \ldots, 1)^t$. Here, however we have an inequality).

Exercise 1.2.16 A matrix is *totally unimodular* if any square submatrix has determinant $-1, 0$ or 1. Prove that the incidence matrix of a bipartite graph is *totally unimodular* ([1, Exercise 4.2.4]).

Exercise 1.2.17 Prove that in case of a bipartite graph the feasible region of the LP relaxation of the Integer Linear Program in Exercise 1.2.15 has all its vertices in $\mathbb{N}^{|E(G)|}$. (See [1, Theorem 8.28]).

Exercise 1.2.18 Let $G[X, Y]$ be a bipartite graph. In Exercise 1.2.15 we phrased a maximum matching problem as an Integer Linear Program and we have proved with the Exercises above that we can find find an optimal solution via linear programming.

1. State an Integer Programming problem for finding a minimal cover of G.
2. Consider the LP relaxation. Is the feasible region a lattice polytope?
3. What is the dual of the LP problem?

Exercise 1.2.19 (Proof of Theorem 1.2.14) Combine the exercises above to get a proof of Theorem 1.2.14.

1.2.2 Hall's Theorem

For a moment assume that G is bipartite with bipartition $V = X \cup Y$. We answer the question of when we can find a matching M, matching all vertices of X. Recall from Graph Theory 1, that for S a subset of the vertices of a graph, $N(S)$ denotes the set of neighbouring vertices of vertices in S.

Theorem 1.2.20 (Hall's Marriage Theorem) *Let $G[X, Y]$ be a bipartite graph. This graph has a matching, matching all vertices of X if and only if*

$$\forall S \subseteq X : |N(S)| \geq |S|.$$

Proof. $\Rightarrow$: Let $S \subseteq X$ and M a maximum matching. Because all vertices of S are matched in M, we must have $|N(S)| \geq |S|$ as desired.

$\Leftarrow$: Suppose that $\forall S \subseteq X : |N(S)| \geq |S|$, but that it was not possible to match all vertices of X. Let M^* be a maximum matching. Then $|M^*| < |X|$. Let $u \in X$ be a non-matched vertex. Let

$$Z := \{v \in V(G) : \exists M^*\text{-alternating path from } u \text{ to } v\}.$$

Let $R := Z \cap X$ and $B := Z \cap Y$. Every vertex in $R \setminus \{u\}$ is matched in M^* to a vertex of B. Moreover, every vertex in B is matched in M^* to a vertex of R (if some vertex $v \in B$ was not matched, there would be an augmenting path from u to v leading to a matching with more edges than $|M^*|$ according to Berge's theorem. That would be a contradiction). Therefore $|B| = |R| - 1$. By the definition of Z we also have that $N(R) = B$. We conclude $|N(R)| = |B| = |R| - 1 < |R|$ contradicting $\forall S \subseteq X : |N(S)| \geq |S|$. $\square$

1.3 Augmenting path search

We are again considering a graph G and want to find a maximum matching. Suppose we have some matching M in G. Our goal will be to

- find an M-augmenting path
- or prove that none exists.

This is desirable since if we find an M-augmenting path P, then the matching M can be improved by letting $M := M \triangle P$. Conversely, Berge's Theorem 1.2.3 says that if no augmenting path exist then M is maximum. Thus we want to search for an augmenting path i.e. do an "Augmenting Path Search". The search starts at some vertex u and our first objective is to find an M-augmenting path starting here (although we do not always succeed). During the search, we build up an *M-alternating tree* with root u:

Definition 1.3.1 Let $G = (V, E)$ be a graph, M a matching in G and $u \in V$. A subtree T of G rooted at u satisfying:

$$\forall v \in T : uTv \text{ is } M\text{-alternating}^4$$

is called an *M-alternating* u-tree. The M-alternating tree is called *M-covered* if every vertex other than u is covered by an edge in $M \cap T$.

Example 1.3.2 In Figure 2, deleting any of the edges in the three-cycle, we obtain an M-alternating u-tree for any choice of u and M being the bold edges. Choosing u as the M-uncovered vertex, two of the three trees are M-covered.

[4]Notation: If T is a tree with vertices u and v then uTv denotes the unique path from u to v in T. We later see an expression $uTxey$ where e is an edge between $x \in V(T)$ and y. This denotes the path in T between u and x, concatenated with the edge e, giving a path ending at y.

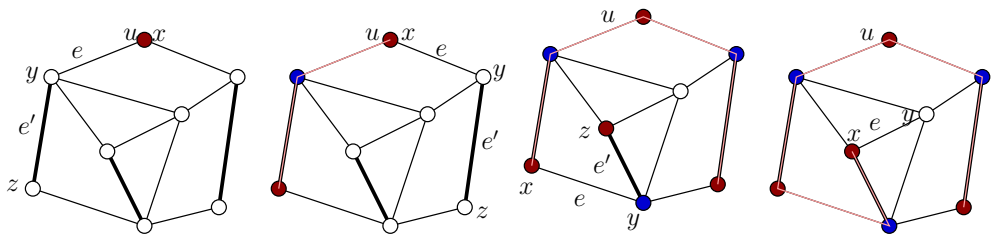

Figure 3: A run of the APS Algorithm 1.3.3 where an M-augmenting path of length 5 from u to y is found while we build up a pink M-alternating u-tree.

Let u be an M-uncovered vertex. The idea now is to start with the empty M-alternating tree with root u and expand it as we search for an augmenting path starting at u, see Figure 3. We colour the tree such that every vertex with an even distance in T to u is red and every vertex with an odd distance is blue.

We want to find

- (preferably) an M-augmenting path
- or a maximal M-covered u-tree.

By a maximal M-covered u-tree we mean a tree that cannot be grown further an still by an M-covered u-tree. In some cases (for example the bipartite case) the maximal tree will help us argue about non-existence of M-augmenting paths.

Algorithm 1.3.3 (Augmenting Path Search) [5]
Input: *A graph G with a matching M and an unmatched vertex $u \in V(G)$.*
Output: *An M-augmenting path in G or a maximal M-covered u-tree in G*

- Let T be the tree with only one vertex u and no edges.
- Red $:= \{u\}$, Blue $:= \emptyset$.
- while($\exists x \in$ Red and $e = (x, y) \in E(G) : y \notin V(T)$)
  - If y is not M-covered
    - Return the M-augmenting path $uTxey$.

    else
    - Let $e' = (y, z)$ be an edge in M.
    - Add y, z, e and e' to T.
    - Blue $:=$ Blue $\cup \{y\}$.
    - Red $:=$ Red $\cup \{z\}$.
- Return T.

[5] About "return" statements: In our notation reaching a "return" statement also implies that the algorithm stops. That is, if in some iteration of the loop the first if-statement is satisfied, the algorithm is interrupted after a (single) M-augmenting path is returned.

Proof. The while condition checks if T is a maximal M-alternating tree in G. If the tree can be extended with a single unmatched vertex y (and edge), the desired M-augmenting path can be returned. If not, we can grow the tree. The only thing to observe in the second step is that z is not already in the tree, since all vertices of T (except u which is not M-covered) are already matched by another vertex of the tree, but z is matched y which is not in the tree. The algorithm terminates because T grows in each iteration and G is finite. □

Exercise 1.3.4 What is "a polynomial time algorithm"? Argue that Algorithm 1.3.3 is a polynomial time algorithm.

Remark 1.3.5 Every u-tree implicitly carries a colouring of its vertices. Namely, u is coloured red and all other vertices are alternatingly coloured blue and red by starting from the root and going towards the leaves.

The following proposition enables us to argue about non-existence of M-augmenting paths.

Proposition 1.3.6 *Let $u \in V(G)$ be uncovered (unmatched) by M. If G has a maximal M-covered u-tree T such that*

- *no red vertices are adjacent in G to a vertex in $V(G) \setminus V(T)$*
- *and no two red vertices of T are adjacent in G*

then G has no M-augmenting path containing a vertex from T.

Proof. Suppose an M-augmenting path P in G existed and that it contained some vertex from T.

- Some vertex $v \neq u$ must be involved in both $V(P) \cap V(T)$. (Because u is not covered by M, u is an end of P. By the first assumption the vertex after u in P is also in $V(T)$.)
- Because T is alternating, v is covered by an edge $e \in M \cap E(T)$.
- Let's follow P starting at e going down the tree.
- If P branches off T, it must happen at a red vertex. (The edge in P following a blue vertex b must be the edge from M covering b in T.)
- Suppose P leaves T at a red vertex $x \in T$ to a vertex y. Then y must be in some other part of the tree (by the first assumption) and y must be blue (by the second assumption). The blue vertex y cannot be the end of the augmenting path since it is M-covered. The edge following y in P is M-covered and y is matched with a red vertex in T. Therefore the path must follow the tree (away from the root) from y to its next vertex.
- Continuing this way we follow P and see that P has no chance of reaching its end. This is a contradiction since P is finite.

We conclude that no such augmenting path exists. □

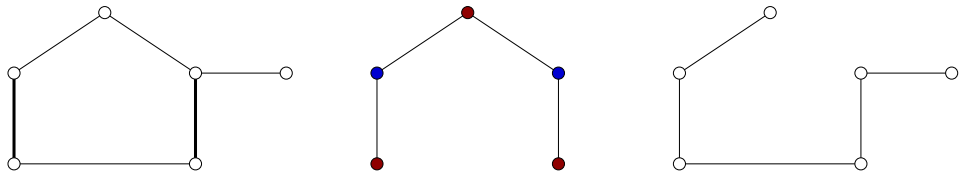

Figure 4: The graph G in Example 1.3.7 with its matching M, a maximal M-covered tree in G, and an augmenting path.

For simplicity we may call a maximal M-covered u-tree (rooted at an M-uncovered vertex) computed by the APS Algorithm 1.3.3 an APS-tree. An APS-tree does not always satisfy the second assumption of Proposition 1.3.6 above. That makes it difficult to use the tree as the following example shows.

Example 1.3.7 Consider the graph G shown to the left in Figure 4 with an associated matching M and an M-uncovered vertex u (shown on the top of the figure). If we compute an APS-tree using Algorithm 1.3.3 starting at u, we may get the tree T in the middle of the figure. This tree does not satisfy the second assumption of Proposition 1.3.6. Moreover, the matching number of $\alpha'(G)$ is 3. An augmenting path is shown to the right.

However, if G is **bipartite**, then the two assumptions in the Proposition are satisfied for the Augmenting Path Search output and we have:

Corollary 1.3.8 *Let G be a bipartite graph and M a matching and u a vertex not covered by M. If Algorithm 1.3.3 produces a tree T (that is, it does not produce an augmenting path) then there exists no augmenting path containing any vertex from T.*

Proof. Because the algorithm only returns T when the "while" condition fails, the tree T must satisfy the first condition of Proposition 1.3.6. Because the graph is bipartite, the second condition is also met. The conclusion follows. □

To find a maximum matching in a bipartite graph we could, of course, start with $M = \emptyset$ and then apply the APS Algorithm 1.3.3 and the corollary for all M-uncovered vertices u. Either we conclude that no M-augmenting path exists - showing that M is maximum - or we find an M-augmenting P and do the "$M := M \triangle P$" operation to improve the matching. Repeating this process we will end with a maximum matching. However, if we have already once found a maximal M-alternating tree T rooted at u, we can avoid considering all vertices involved in T again later in the computation. This is explained by the recursive procedure below.

Algorithm 1.3.9 (Egerváry's Algorithm) [6]
Input: *A bipartite graph $G[L, R]$ with some matching M.*
Output: *A maximum matching M^* and minimum covering X^* of G.*

[6]The ideas of Algorithm 1.3.9 can be implemented more efficiently as the Hopcroft-Karp Algorithm, which performs better in theory and practise.

- *If $E(G) = \emptyset$ then return $(M^*, X^*) := (\emptyset, \emptyset)$.*
- *While(there exists an M-uncovered vertex $u \in V(G)$ and Algorithm 1.3.3 run on (G, M, u) produces an augmenting path P)*
  - $M := M \triangle P$.
- *If G has no M-uncovered vertex, return $(M^*, X^*) := (M, L)$.*
- *Let T be the tree produced by Algorithm 1.3.3 and B its blue vertices.*
- *Let $G' = G \setminus T$ and $M' = M \setminus T$*
- *Recursively produce a maximum matching M'' and minimum covering X'' of G' by calling Egerváry's algorithm on (G', M').*
- *Return $(M^*, X^*) := ((T \cap M) \cup M'', X'' \cup B)$.*

Before reading the proof, try to read Example 1.3.10 and do Exercise 1.3.11 below.

Proof. Since the graph is finite, the while-loop cannot go on forever. Therefore T will be produced with at least one vertex. When calling recursively, it therefore happens on a graph with fewer vertices. This proves termination.

Notice that assuming $|M''| = |X''|$ we get $|(T \cap M) \cup M''| = |(T \cap M)| + |M''| = |B| + |X''| = |B \cup X''|$. Therefore the algorithm always produces a pair (M^*, X^*) with $|M^*| = |X^*|$.

We claim that the produced M^* is a matching. This follows from M'' being a subset of the edges of $G \setminus T$.

Notice that in G there cannot be any edges between vertices from red vertices of T to vertices from $G \setminus T$ (this follows from the proof of Corollary 1.3.8 as the proof of the first condition of Proposition 1.3.6). Therefore vertices of B cover all edges incident to vertices of T. Finally, the edges of $G \setminus T$ are covered by X''. Therefore $X^* = B \cup X''$ covers G.

By Lemma 1.2.11 any matching has size at most $|X^*|$. Because $|M^*| = |X^*|$ we get that M^* is maximum and therefore $|M^*| = \alpha'(G)$. Similarly, by Lemma 1.2.11 any covering has size at least $|M^*|$. Because $|X^*| = |M^*|$ we get that X^* is minimum and therefore $|X^*| = \beta(G)$. □

Example 1.3.10 Consider the graph G in Figure 5 with a matching M of size 2. The APS algorithm might give us the maximal M-covered tree T in the second picture. By Proposition 1.3.6, taking symmetric difference with M-augmenting paths cannot affect T. Therefore we may ignore T when searching for augmenting paths and restrict to the graph $G \setminus T$ shown to the right.

Notice that no red vertex is connected to a vertex of $G \setminus T$. Therefore a covering of $G \setminus T$ together with the blue vertices is a covering of G. Finding a maximal matching M'' in $G \setminus T$ with the same size as its minimal covering, we get a matching $(M \cap T) \cup M''$ of G.

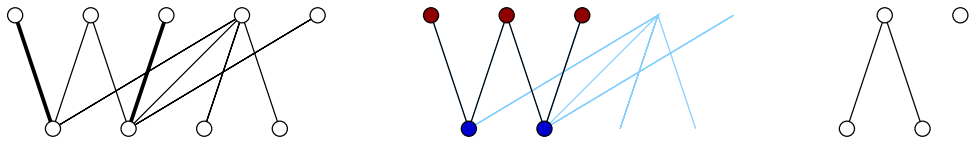

Figure 5: The graph G in Example 1.3.10 with its matching M, a maximal M-covered tree T in G, and the graph $G \setminus T$.

Exercise 1.3.11 Run Algorithm 1.3.9 on the graph G in Figure 5, starting with $M = \emptyset$ and choosing u to be the first vertex in the first iteration, the third vertex in the second iteration, and the second vertex in the third iteration. In the recursive call choose u to be the fourth vertex (from the upper left). What is the computed minimum covering for G?

Remark 1.3.12 According to the proof of Algorithm 1.3.9, the algorithm always produces a matching and a covering of the same size. Hence we now have an alternative proof of Theorem 1.2.14, which we first proved in Exercise 1.2.19.

Exercise 1.3.13 Is Algorithm 1.3.9 a polynomial time algorithm?

1.4 Matchings in general graphs

In this section we aim at finding maximum matchings in general (non-bipartite) graphs.

Exercise 1.4.1 Is the incidence matrix of a (not necessarily bipartite) graph always totally unimodular? Can you find a graph G where the LP relaxation of the Integer Linear Program of Exercise 1.2.15 does not have an optimal solution in $\mathbb{N}^{|E(G)|}$?

The exercise shows that for general graphs we cannot just solve the Integer Linear program by considering the relaxation. However, a theorem by Edmonds (which we might see later) tells us which linear inequalities to add to the ILP to make the feasible region of the LP relaxation a lattice polytope.

1.4.1 Certificates

For a graph G with a matching M, if we can find a covering with X with $|X| = |M|$, then Proposition 1.2.12 implies that $\alpha'(G) \geq |M| = |X| \geq \beta(G) \geq \alpha'(G)$. This shows that the matching is maximum and the covering is minimum.

For a bipartite graph the König-Egerváry Theorem states that the matching number $\alpha'(G)$ equals the covering number $\beta(G)$. In particular if we have a maximum matching M, there will always exist a covering X of the same size proving that M is maximum. Such a proof X we also call a *certificate*. Egerváry's Algorithm 1.3.9 produces a certificate for the found matching being maximum.

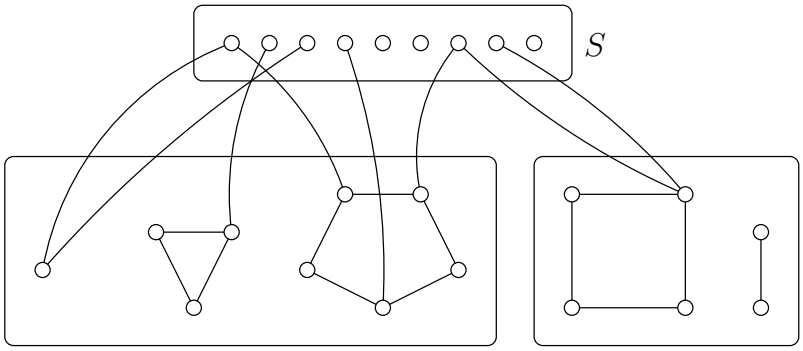

Figure 6: How the vertices of a graph G are split into three categories for a chosen subset of vertices S, being the 9 vertices on the top.

Example 1.4.2 The complete graph K_3 on three vertices with edges $e_1, \ldots, e_3$ has $\alpha'(K_3) = 1 < 2 = \beta(K_3)$. This proves that the maximum matching $M = \{e_1\}$ has no certificate in form of a covering because such a covering will have size at least 2. That no certificate exists can happen because K_3 is not bipartite.

The example shows that we need another kind of certificate when the graph is not known to be bipartite.

Let a graph G with a matching M be given. Define $U \subseteq V(G)$ to be the set of vertices not covered by M. Let $S \subseteq V(G)$ be any vertex subset. We now imagine drawing the graph as in Figure 6. There the vertices of G have been arranged into three categories: vertices in S, vertices belonging to odd components of $G \setminus S$ and vertices belonging to even components of $G \setminus S$.

Each odd component must have one vertex not covered by M — or rather that is unless that vertex was matched with a vertex in S. At most $|S|$ odd components can be "saved" in this way. Therefore

$$|U| \geq o(G \setminus S) - |S| \tag{1}$$

where $o(H)$ for a graph H is the number of odd components of H.

For the graph in Figure 6 this bound is useless because the set S was chosen too big. Luckily, we get a bound for every choice of subset. If S is chosen cleverly it can be used to argue that a given matching is maximum.

Example 1.4.3 Consider the graph G in Figure 7 and some matching M with corresponding uncovered vertex set U. Choosing S to be the center vertex, Inequality 1 becomes

$$|U| \geq o(G \setminus S) - |S| = 3 - 1 = 2.$$

Hence for any matching M, there will always be two vertices that are not covered by M. Therefore the size of a matching M is at most $(10 - 2)/2 = 4$.

From the example we see that the lower bound on the number of uncovered vertices can be turned into an upper bound on the size of a matching. In

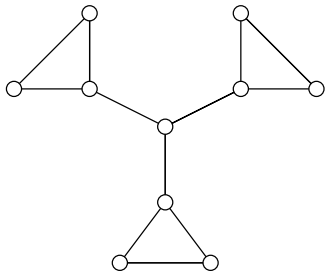

Figure 7: The graph of Example 1.4.3.

particular, if we for our matching M can find a set $S \subseteq V(G)$ such that

$$|V(G)| - 2|M| = o(G \setminus S) - |S|. \tag{2}$$

then we know that M is maximum. A set $S \subseteq V(G)$ is called a *barrier* if there exists a matching M such that Equation 2 is satisfied. A barrier can be used as a certificate, proving that a given maximum matching M is indeed maximum.

Exercise 1.4.4 What is a barrier for the 4-vertex graph consisting of a path of length 2 (with three vertices) and an isolated vertex?

Exercise 1.4.5 Find a barrier for the graph in Figure 5. Prove that every bipartite graph has a barrier.

That every graph has a barrier can be proved by a version of Edmond's Blossom Algorithm 1.4.8.

1.4.2 Edmonds' Blossom Algorithm

Recall that the problem with having a non-bipartite graph is that the augmenting path search (Algorithm 1.3.3) might, as in Example 1.3.7, return a tree with two adjacent red vertices. Consequently, Proposition 1.3.6 cannot be applied and we cannot exclude the existence of an augmenting path.

Let T be a M-covered u-tree coloured red and blue according to the conventions for the augmenting path search. Let x and y be two red vertices both incident to an edge e. The graph $T \cup \{e\}$ has a unique cycle C. This cycle has odd length and is called a *blossom*. Every vertex of C except one is covered by $M \cap E(xTy)$.[7] Let r denote this uncovered vertex.

The idea of Edmonds' blossom algorithm is to let the Augmenting Path Search detect (when colouring red) adjacent red vertices in the tree. When such a pair of vertices is detected, the APS algorithm will be terminated, returning a blossom C. The algorithm proceeds searching for an augmenting path in G/C.[8]

[7] By xTy we mean the unique path in the tree T with ends x and y.

[8] Recall that a for a loop-less graph G and a set $S \subseteq V(G)$, the contracted graph G/S is obtained by replacing all vertices of S by a single new vertex. We say that the vertices are *identified*. The edge set remains the same, except that possible loops are removed. The notation G/S is not to be confused with $G \setminus S$ (where the vertices of S are completely deleted from the graph).

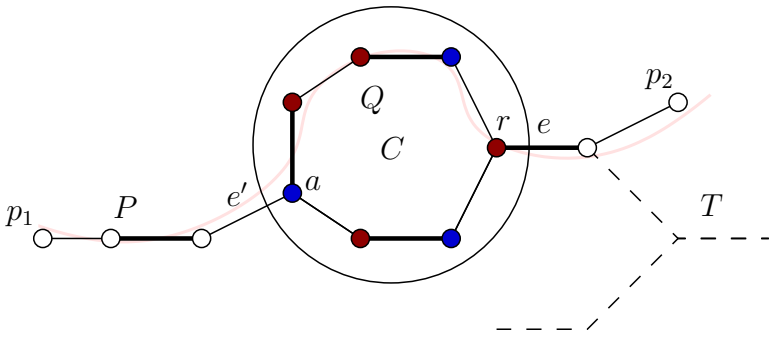

Figure 8: The situation in the last paragraph of the proof of Lemma 1.4.6.

If an augmenting path P is found in G/C, then that path can be lifted to an augmenting path P' of G. Hence the matching can be improved by forming $M \triangle P'$.

When identifying $V(C)$ we give the new vertex the name R. A matching M in G induces a matching in G/C with $(|C|-1)/2$ fewer edges. This matching we also denote by M. Note also that a blossom is not just an alternating cycle of odd length - for the proof of Lemma 1.4.7 below it is important that $r \in G$ is connected to some uncovered vertex (u) by an alternating path.

Lemma 1.4.6 *Let G be a graph with a matching M. Let u be an M-uncovered vertex and T an M-alternating u-tree. Let C be a blossom with $(E(C) \cap M)$-uncovered vertex r. If G/C has an augmenting path P then G also has an augmenting path.*

Proof. If $R \notin V(P)$, then P is M-augmenting in G.

If P in G/C ends in R with the last edge being $e = (a, R)$, then there must be an edge $e' = (a, s) \notin M$ where $s \in C$. In C there is an alternating (s, r)-path Q starting with an edge in M. Because R is not M-covered in G/C, r is not M-covered in G. Therefore $PsQr$ is augmenting in G.

If P in G/C passes through R, then some edge $e \in M$ is incident to r in G. Let $e' \notin M$ be the other edge in P incident to R in G/C. Starting in C at the end a of e' in G there is an alternating path Q from a to r starting with an edge of M. The concatenated path $p_1PaQrPp_2$ is now M-augmenting in G, where p_1 and p_2 are the ends of P. (See Figure 8.) □

The following lemma is more tricky. We somehow need to argue that even if we for example collapse an edge in M from an augmenting path P in G when going from G to G/C, an augmenting path in G/C can still be found.

Lemma 1.4.7 *Let G be a graph with a matching M. Let u be an M-uncovered vertex and T an M-alternating u-tree. Let C be a blossom with $(E(xTy) \cap M)$-uncovered vertex r. If G has an augmenting path P then G/C also has an augmenting path.*

Proof. If $V(P) \cap V(C) = \emptyset$ then we can just take the same path in G/C.

Therefore we now assume that $V(P) \cap V(C) \neq \emptyset$. Clearly, P cannot be contained in C because P has two M-uncovered vertices. Let Q be the subpath of P starting at one end of P until C is reached. That is, Q has exactly one vertex in $V(C)$.

If $r = u$ then r is not covered and there is no $e \in M$ going into C. Hence the last edge of Q is not in M. Since the new vertex R in G/C is not M-covered, Q is M-alternating in G/C.

If $r \neq u$, let $S := uTr$. We now apply Berge's Theorem 1.2.3 four times. First, by this theorem the existence of an M-augmenting path in G implies that M is not a maximum matching. Because $|S \triangle M| = |M|$, also $S \triangle P$ is not maximum. By Berge's Theorem G has an $S \triangle M$-augmenting path. For the matching $S \triangle M$ we could start the APS algorithm at the uncovered vertex r and obtain again the blossom C. The argument of the $r = u$ case therefore implies that G/C has an $S \triangle M$-augmenting path. By Berge Theorem, $S \triangle M$ is not maximum in G/C. Because M and $S \triangle M$ have the same size as matchings in G/C, M is not maximum in G/C. By Berge Theorem, the desired M-augmenting path in G/C exists. □

Algorithm 1.4.8 (Edmonds' Blossom Algorithm)
Input: *A graph G with some matching M.*
Output: *An M-augmenting path in G. Or, if M is maximum, output $\emptyset$.*

- *If all vertices $V(G)$ are M-covered, then return $\emptyset$.*
- *Choose an M-uncovered vertex $u \in V(G)$.*
- *Call the Augmenting Path Search (Algorithm 1.3.3) on G and u.*
- *There are now three cases:*
  - *If APS found an augmenting path, then return it.*
  - *If APS produced an M-covered u-tree T with two red vertices adjacent in G, then*
    - *Choose two adjacent two red vertices and let C denote the associated blossom.*
    - *Recursively*[9] *try to compute an M-augmenting path in G/C.*
    - *If successful, lift the path to an M-augmenting path in G following the construction in the proof of Lemma 1.4.6 and return it.*
    - *If unsuccessful, return $\emptyset$.*
  - *If there are no red vertices in T being adjacent in G then*
    - *Let $G' = G \setminus T$ and $M' = M \setminus T$*
    - *Recursively compute an M'-augmenting path in G'*
    - *If successful return the path.*

[9] An algorithm is called *recursive* if it calls itself. When we write "recursively compute" we mean do the smaller computation with the algorithm itself.

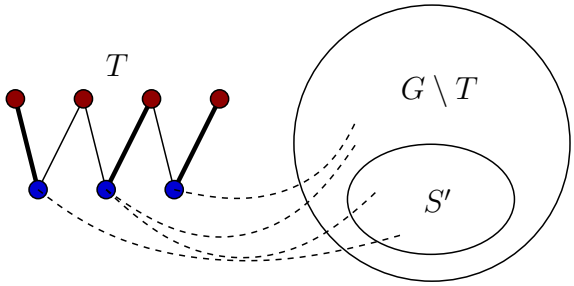

Figure 9: Figure showing how a the barrier of the smaller graph is combined with the blue vertices of the bigger graph to produce a barrier for the bigger graph as explained in the last step of the proof of Algorithm 1.4.8.

∗ *If unsuccessful, return* $\emptyset$.

Ideally the algorithm should should return a barrier as a certificate of optimality of the matching. Unfortunately our version does not.

Proof. We first observe that the algorithm terminates because the recursive calls are always done on graphs with fewer vertices.

We will recursive/inductively prove correctness.There are three cases to consider:

- In the base case, where there are no uncovered vertices, it is correct not to return a path.
- If an augmenting path is found it is correct to return it.
- If no adjacent red vertices exist, then Proposition 1.3.6 says that it is correct to restrict the search to the complement of T.
- If a blossom was found, then Lemma 1.4.6 and Lemma 1.4.7 tell us that it is correct to look for an augmenting path in G/C.

□

Remark 1.4.9 In the spirit of Berge's Theorem 1.2.3 we may, similarly to Egerváry's Algorithm 1.3.9, start with some matching M in a graph G, and enlarge it by taking symmetric difference $M := M \triangle P$ repeatedly where P is an M-augmenting path found by the blossom algorithm.

Exercise 1.4.10 Use Edmonds' Blossom shrinking Algorithm 1.4.8 to find a maximum matching (and a barrier?) for the two graphs in Figure 10.

Exercise 1.4.11 Prove that Edmonds' Blossom Algorithm (Algorithm 1.4.8) is a polynomial time algorithm.

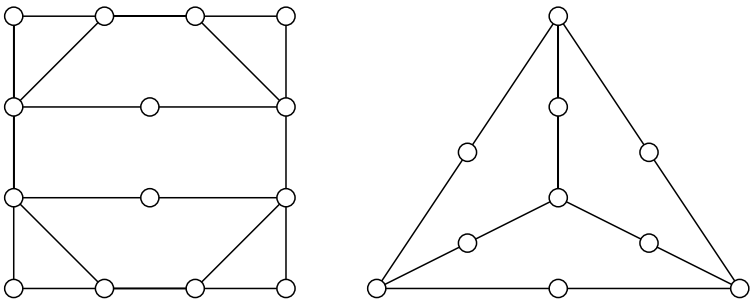

Figure 10: The graphs in Exercise 1.4.10.

Exercise 1.4.12 Can we prove Lemma 1.4.7 using the algorithm? (That is, can we change the proof of correctness, so that it does not rely on Lemma 1.4.7?)

Remark 1.4.13 Notice that we may change the APS algorithm so that it returns a (not necessarily maximal) covered tree as soon as two adjacent red vertices are discovered. This will speed up the computation in Algorithm 1.4.8.

Exercise 1.4.14 A naive way to compute a maximum matching is to apply Algorithm 1.4.8 repeatedly. By changing the algorithm, it is be possible to avoid computing the same maximal covered trees repeatedly. How?

1.5 Maximum weight matching in bipartite graphs

We now let $G[X, Y]$ be a bipartite graph with a weight function $w : E \to \mathbb{R}$. The weight of a matching M (or any other subgraph of G) is defined as $w(M) = \sum_{e \in M} w(e)$. We will see how the idea from Egervary's algorithm can be used to find matchings in G with largest weight.

We first discuss the case where G is a complete bipartite graph and we want a maximal weight perfect matching.[10]

The algorithm below computes iteratively a matching M_1 in G of size 1 with largest weight, a matching M_2 of size 2 and so on until the maximum w-weight perfect matching is found. In this process it is *possible to go from M_{i-1} to M_i by taking the symmetric difference with an augmenting path.* This is a consequence of Lemma 1.5.2 below.

We first introduce some notation. For a matching M we define the directed graph G_M to be G with the edges orientation from X to Y, with the exception that edges of M are oriented from Y to X. We define the weight function w_M on G_M by letting $w_M(e) = w(e)$ if $e \in M$ and $w_M(e) = -w(e)$ otherwise.

Example 1.5.1 Consider $G = K_{2,2}$ with weights as in Figure 11. A maximal weight matching M_1 of size 1 is shown in the second picture. From this the graph G_{M_1} with weights w_{M_1} is formed. In the fourth picture a max-weight

[10] In general a bipartite graph $G[X, Y]$ is *complete* if any two vertices $x \in X$ and $y \in Y$ are adjacent. We use the notation $K_{r,s}$ to denote the complete bipartite graph with bipartition (X, Y) where $|X| = r$ and $|Y| = s$ (and there are no parallel edges).

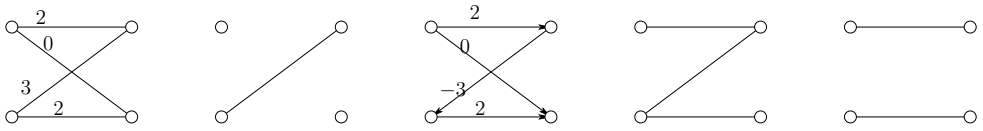

Figure 11: The graph of Example 1.5.1. The pictures show G with weights w, M_1, G_{M_1} with weights w_{M_1}, P_1 and M_2, respectively.

path P_1 from an M_1-uncovered vertex in X to an M_1-uncovered vertex in Y is shown. Taking symmetric difference $M_2 := M_1 \triangle P_1$ we get the max-weight perfect matching M_2.

Observe that

$$w(M \triangle P) = w_M(P) + w(M). \tag{3}$$

Lemma 1.5.2 *Let M be a matching in G with $|M| = d$. Suppose M has maximal w-weight among all matchings of size d. Let M' be a matching with $|M'| = d+1$ Then there exists an M-augmenting path P in G such that $M \triangle P$ has size $d+1$ and $w(M \triangle P) \geq w(M')$.*

Proof. Because the degree of any vertex in the subgraphs M and M' of G is at most 1, the symmetric difference $M \triangle M'$ has only vertices with degree at most two. The edge set $E(M \triangle M')$ is therefore a disjoint union of cycles and paths. Since $|M| < |M'|$ one component P of $M \triangle M'$ contains more edges from M' than from M. Because P is M and M'-alternating it cannot be a cycle. Because P contains more edges from M' than from M it is a path of odd length. Moreover, the ends of P are not M-covered. (If they were, the component of $M \triangle M'$ containing P would be larger than P.) Notice that $w_M(P) = -w_{M'}(P)$ because M and M' involve opposite edges along P. Now

$$w(M') = w(M' \triangle P) - w_{M'}(P) \leq w(M) - w_{M'}(P) = w(M) + w_M(P) = w(M \triangle P).$$

The first and last equality follow from Equation 3 above. The inequality follows because M has maximal weight among all matchings of its size – in particular it has w-weight greater than $M' \triangle P$. □

Algorithm 1.5.3 *(Hungarian Algorithm)*
Input: *A complete bipartite graph $G[X,Y] = K_{D,D}$*[11] *and a weight function w.*
Output: *A perfect matching M in G with $w(M)$ maximal.*

- *Let $M = \emptyset$.*
- *While $|M| \neq |X|$*
  - *Form the graph G_M with weight function w_M described above.*

[11] Recall that $K_{D,D}$ is the complete bipartite graph $G[X,Y]$ on $2 \times D$ vertices, meaning that $|X| = |Y| = D$ and any two vertices $x \in X$ and $y \in Y$ form an edge (x,y).

- *Find a maximum w_M-weight directed path P in G_M from an uncovered vertex of X to an uncovered vertex of Y.*
- *Let $M := M \triangle P$.*

- *Return M*

Proof. The algorithm terminates because M increases by 1 every time the symmetric difference with the M-augmenting path P is taken.

To prove correctness, we prove by induction that the following claim holds:

S_i: After each the ith iteration, M has maximal possible w-weight among all matchings of size i

Base case $i = 0$: After 0 iterations, $M = \emptyset$. This is the only matching of size 0 and therefore the statement S_0 is satisfied.

Induction step: Suppose S_{i-1} is true. We wish to prove S_i. By Lemma 1.5.2 there exists some M-augmenting path P such that $w(M \triangle P)$ is maximal among weights of all matchings of size $i = |M| + 1$. Because $w(M \triangle P) = w(M) + w_M(P)$ for every M-augmenting P, we can simply choose an M-augmenting path P from X to Y such that $w_M(P)$ is maximal. Now $w(M \triangle P)$ is maximal of size i. Therefore S_i is true.

By induction we get that S_i is true for all $i \leq |X|$. In particular for $i = |X|$ we get that $w(M)$ is maximal among the weights of all perfect matchings. □

Example 1.5.4 Consider the graph $K_{3,3}$ with the weighting w given in Figure 12. In three iterations the max-weight matching M_3 is computed. The values of G_M, w_M and P are shown at each iteration to the right of the figure. At the last step, there actually are two choices of P_2 both with weight 2. We choose one of them.

We end this section with an observation that will be useful later:

Lemma 1.5.5 *In Algorithm 1.5.3, the graph G_M with weighting w_M does not have any directed cycle with positive weight.*

Proof. Suppose such a cycle C did exists. Then consider $M \triangle C$, which is a matching of the same size as M. We have $w(M \triangle C) = w(M) + w_M(C) > w(M)$. This contradicts M having largest w-weight among matchings of its size. □

Exercise 1.5.6 Let $a \in \mathbb{N}$ and $K_{d,d}$ have weights given by a weight function w. How would you compute a matching of maximum weight among all matchings of size a?

Exercise 1.5.7 How do we find the maximum weight (not necessarily perfect) matching in a complete bipartite graph $K_{d,d}$?

Exercise 1.5.8 How do we find the maximum weight matching in a bipartite graph (not necessarily complete)?

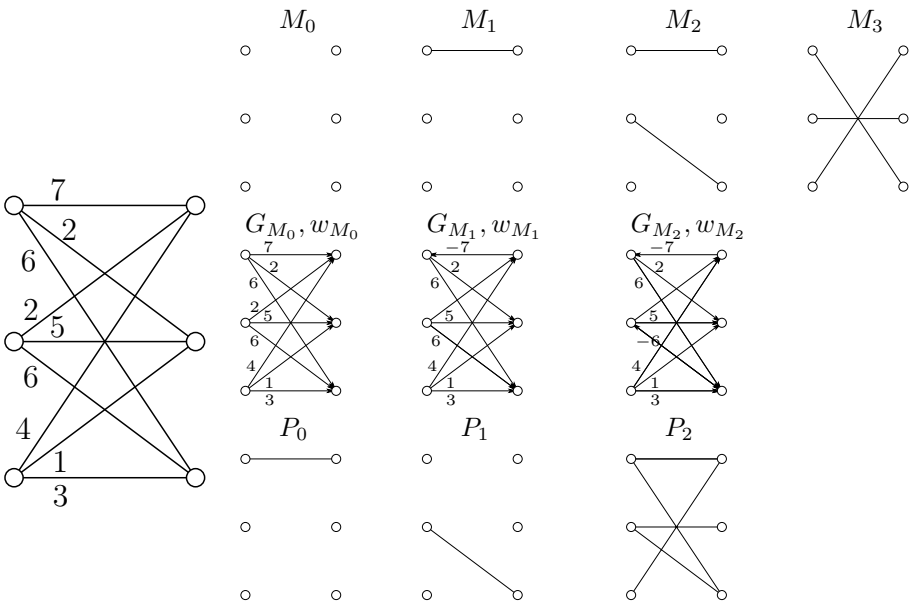

Figure 12: The graph of Example 1.5.4 with the values of M, G_M, w_M and P through a run of the Hungarian Algorithm 1.5.3.

Exercise 1.5.9 Let a bipartite graph with weights be given. (Let's just assume the weights are non-negative). Among all matchings of largest size, how do we find the one with largest weight?

Exercise 1.5.10 The Hungarian method has many disguises. Actually, the way it is presented here is not typical. For a different presentation watch the video:

`http://www.wikihow.com/Use-the-Hungarian-Algorithm`

Find a minimum weight perfect matching for the example in that video using Algorithm 1.5.3. Find a maximum weight perfect matching of Example 1.5.4 using the algorithm of the video. (Which theorem was used at 2:35?)

The two descriptions are not identical, and it is not obvious that they are more or less the same. An explanation why it is fair to also call Algorithm 1.5.3 the "Hungarian method" is given in [7, end of Chapter 8].

1.6 Distances in graphs (parts of this section are skipped)

Motivated by Algorithm 1.5.3 (and in particular the $K_{4,4}$ example in Exercise 1.5.10) we need a method for computing longest/shortest paths in a graph between two sets of vertices. We will assume that our graphs are directed.

From the course *Mathematical Programming* we know Dijkstra's algorithm. What was known as weights in Section 1.5 is now called lengths or distances. Dijkstra's algorithm computes the shortest distance between two given vertices in a graph with edges of positive length:

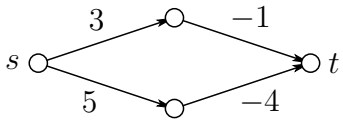

Figure 13: Dijkstra's Algorithm will fail on this graph. Starting at s, the algorithm first finds the distance 3 to the upper vertex and thereafter concludes that the distance to t is 2. Only later when the edge with length 5 has been processed it realises that t is actually reachable from s with a distance of 1. Had the graph been bigger Dijkstra's algorithm might have used that the false (s,t)-distance 2 for other parts of the graph. Correcting previously computed distances is complicated and is not part of Dijkstra's algorithm.

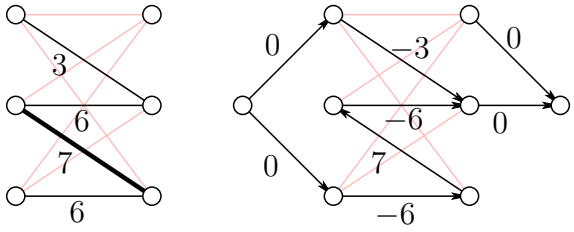

Figure 14: Suppose we want to apply the Hungarian method to find the max-weight matching on the graph on the left, where the pink edges have weight 0. In the first step, the augmenting path consisting of the edge with weight 7 is chosen. Now we want to find a longest path in G_{M_1} with weights w_{M_1}. Or using Dijkstra's algorithm, find the shortest path from the left vertex to the right vertex in the graph to the right. For the same reason as in Figure 13, Dijkstra's algorithm will reach the conclusion that the shortest path is of length -3, when really it is of length -5.

Algorithm 1.6.1 *Dijkstra's Algorithm*
Input: *A directed graph G with a distance function $w : E(G) \to \mathbb{R}_{\geq 0}$ and two vertices s and $t \in V(G)$.*
Output: *An (s,t)-path P with the property that $w(E(P))$ is a small as possible.*

(A more general version of Dijkstra's algorithm takes only one vertex s as input. Distances and shortest paths from s to all other vertices would be returned.)

The problem with having negative weights is illustrated by Figure 13. This problem persists even when the weights satisfy Lemma 1.5.5 as we now show. To solve the problem of finding a largest w-weighted path in Algorithm 1.5.3, extend G_M to a graph G' with new vertices s and t and connected s to all M-uncovered vertices of X and all M-uncovered vertices to t via directed edges. The distances of G' we set to $-w_M$ because we are interested in largest w_M-weighted paths. The additionally introduced edges get length 0. In Figure 14 we apply Dijkstra's Algorithm to the problem, and get the wrong result.

It is not clear if it is possible to repair Dijkstra's Algorithm to also work with

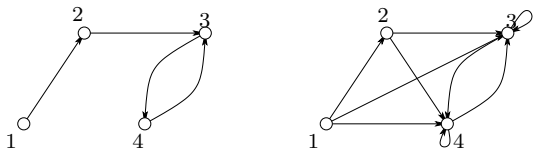

Figure 15: The graphs of Example 1.6.2.

negative weights. In the following subsections discuss how shortest distances in directed graphs can be computed even when weights are negative.

1.6.1 Warshall's Algorithm

If there is no directed (x, y)-path between two vertices x and y in a directed graph G, we say that their distance is ∞. A first step to computing the vertex-vertex distances in a weighted graph would be to figure out which vertices are not reachable from each other. We want to compute the *transitive closure* of G which we now define.

Directed graphs with vertex set V without parallel arcs are in bijection to relations on V. Namely, let G be a directed graph with no parallel edges. Such a graph gives rise to the relation:

$$u \sim_G v \Leftrightarrow (u, v) \in E(G)$$

for $u, v \in V(G)$. The relation $\sim_G$ is called transitive if for $u, v, w \in V(G)$:

$$u \sim_G v \wedge v \sim_G w \Rightarrow u \sim_G w$$

Given a graph G with not parallel edges, its *transitive closure* is the smallest graph G^* containing G such that $\sim_{G^*}$ is transitive.

Example 1.6.2 Consider the graph G on the left in Figure 15. The transitive G^* is shown on the right. After having chosen an ordering on the vertices, we can write the their adjacency matrices. The adjacency matrices for G and G^* are respectively:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

Lemma 1.6.3 *An arc (u, v) is in the transitive closure G^* if and only if there is a directed (u, v)-walk in G of length at least one.*

Proof. Define

$$K := \{(u, v) : \exists \text{ directed } (u, v)\text{-walk in } G \text{ of length at least } 1\}.$$

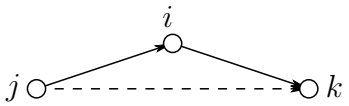

Figure 16: If vertices i, j and k are found in Warshall's Algorithm such that the edge (j, i) and the (i, k) is present, the new edge (j, k) is introduced.

We first prove $K \subseteq E(G^*)$. Let P be a (u, v)-walk in G of length at least one. Let $u_0, \ldots, u_l = v$ be the vertices of P. We have $(u_0, u_1) \in E(G^*)$ and $(u_1, u_2) \in E(G^*)$. Hence $(u_0, u_2) \in E(G^*)$. Because $(u_2, u_3) \in E(G^*)$ also $(u_0, u_3) \in E(G^*)$. Eventually this proves $(u, v) \in E(G^*)$.

Now we prove $E(G^*) \subseteq K$. It is clear that $E(G) \subseteq K$. Notice that the relation on V induced by K is transitive. Since G^* is the smallest graph containing G with $\sim_{G^*}$ transitive, $E(G^*) \subseteq K$. □

Warshall's algorithm, which we now present, computes the transitive closure by observing that if (u, v) and (v, w) are edges in the transitive closure, then so are (u, w). Therefore the algorithm starts with the graph G (which is a subgraph of the transitive closure of G) and expands it to the transitive closure by looking for the pattern in Figure 16. When no such pattern can be found, we have reached the transitive closure. The algorithm is a bit more clever than this - it does not have to restart its search for the pattern after a new edge has been inserted. For convenience, we use names $1, 2, \ldots, |V|$ for the vertices of G.

Algorithm 1.6.4 *Warshall's Algorithm*
Input: *The adjacency matrix M for graph G.*
Output: *The adjacency matrix M' for the transitive closure G^*.*

- *Let $M' = M$.*
- *For $i = 1, 2, \ldots, |V|$*
  - *For $j = 1, 2, \ldots, |V|$*
    - *If ($M'_{ji} = 1$)*
      - *For $k = 1, 2, \ldots, |V|$*
        - *If($M'_{ik} = 1$) then let $M'_{jk} := 1$.*

Proof. Since $|V|$ is constant, the algorithm clearly terminates.

To prove correctness, let $G^0 = G$ and let G^i be the graph represented by M' after the ith iteration of the outer loop.

Clearly we have $G = G^0 \subseteq G^1 \subseteq \cdots \subseteq G^{|V|} \subseteq G^*$ where the last inclusion follows because we only ever add edges which must be in the transitive closure.

To prove that $G^{|V|} \supseteq G^*$, we prove the statement $S_0, \ldots, S_{|V|}$ by induction:

S_i: For $(s, t) \in V \times V$, if there is an (s, t)-walk of length at least 1 with all vertices except s and t being among the i first vertices, then $(s, t) \in E(G^i)$.

In the base case $i = 0$, the statement S_i is that if there is a (s,t)-walk in G involving no other vertices then s and t, then $(s,t) \in G^0 = G$, which is clearly true.

For the induction step, assume S_{i-1}. We want to prove S_i. Let P be an (s,t)-walk of length at least 1 which does not involve other vertices than s, t and the first i. If the ith vertex is not in $V(P)$ then S_{i-1} implies that $(s,t) \in E(G^{i-1}) \subseteq E(G^i)$. On the other hand, if vertex i is in $V(P)$, then we have two cases. If $i \in \{s,t\}$, then there is an (s,t)-walk of length ≥ 1 involving only s,t and vertices among the first $i-1$. By S_{i-1}, $(s,t) \in E(G^{i-1}) \subseteq E(G^i)$. If $i \notin \{s,t\}$ then S_{i-1} and the existence of a subwalk sPi imply that $(s,i) \in E(G^{i-1})$. Similarly $(i,t) \in E(G^{i-1})$. It is in the ith iteration explicitly checked if $(s,i) \in E(G^{i-1})$ and $(i,t) \in E(G^{i-1})$ (when $j = s$ and $k = t$) and in that case the edge (s,t) is added to G^i as desired. Therefore S_i is true.

By induction S_i is true for any i. In particular, when $i = |V(G)|$ it gives the inclusion $G^{|V|} \supseteq G^*$, because by Lemma 1.6.3 $(s,t) \in E(G^*)$ only if there is an (s,t)-walk in G of length at least 1. □

1.6.2 Bellman-Ford-Moore

We now present an algorithm which will produce the shortest distances in a graph even when the weights are negative. We use the words length and weight interchangeably. The only assumption is that the graph has no cycles of negative length. Let $d(u,v)$ denote the minimum w-weight of a directed (u,v)-walk in G. If no such minimum exists we let $d(u,v) = -\infty$ and if no walk exists we let $d(u,v) = \infty$. The following presentation has some resemblance to [11].

In the Bellman-Ford-Moore Algorithm a vertex s is special. We want to find distance from it to all other vertices of G. The idea is to keep a list $\lambda \in (\mathbb{R} \cup \{\infty\})^{|V|}$ of smallest distances discovered so far. If at some point we discover the arc (i,j) with $\lambda_j > \lambda_i + w(i,j)$, then λ_j can be improved.

Algorithm 1.6.5 (Bellman-Ford-Moore)
Input: *A directed graph $G = (V,E)$ with possibly negative lengths $w : E \to \mathbb{R}$, and $s \in V$. The graph G should contain no cycles of negative length.*
Output: *For every $v \in V$ the shortest distance λ_v from s to v.*

- $\lambda := (\infty, \ldots, \infty, 0, \infty, \ldots, \infty) \in (\mathbb{R} \cup \{\infty\})^V$ *with* 0 *being at position* s.
- *while*$(\exists (i,j) \in E : \lambda_j > \lambda_i + w(i,j))$
  - *Let* $\lambda_j := \lambda_i + w(i,j)$

Before we prove termination and correctness of the algorithm, let us see some examples.

Example 1.6.6 Consider the graph of Figure 17. The λ-values for one possible

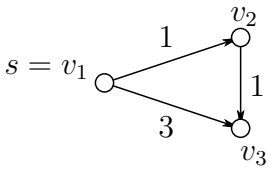

Figure 17: The graph in Example 1.6.6.

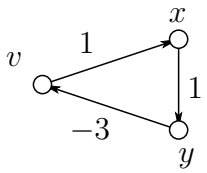

Figure 18: The graph in Example 1.6.7.

run of the algorithm are:

$$\lambda = (0, \infty, \infty)$$
$$\lambda = (0, \infty, 3)$$
$$\lambda = (0, 1, 3)$$
$$\lambda = (0, 1, 2).$$

Example 1.6.7 If the graph contains a cycle with negative length, the Bellman-Ford-Moore Algorithm might not terminate. Consider the graph in Figure 18. We initialise $\lambda_v = 0$ and $\lambda_x = \lambda_y = \infty$. In the first iteration we let $\lambda_x = \lambda_v + 1 = 0 + 1 = 1$. In the next iteration we let $\lambda_y = \lambda_x + 1 = 2$. In the third $\lambda_v = \lambda_y - 3 = -1$. But now we can repeat forever.

Exercise 1.6.8 Consider the graph in Figure 19 with its weighting. Run the Bellman-Ford-Moore Algorithm 1.6.5 on it. Prove that it is possible for the algorithm to update the λ-value of the vertex furthest to the right 32 times in the while-loop. Because of examples like this the Bellman-Ford-Moore algorithm is an exponential time algorithm.

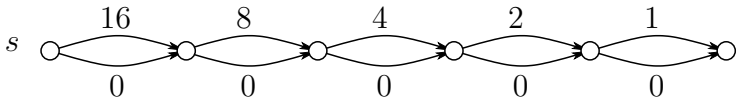

Figure 19: The graph in Exercise 1.6.8.

Lemma 1.6.9 *Suppose G has no directed cycles of negative length (reachable from s). At any time of the algorithm we have*

$$\lambda_v \neq \infty \Rightarrow \exists \text{ a directed path from } s \text{ to } v \text{ of length } \lambda_v.$$

Proof. Find the edge (v', v) leading to the assignment of value λ_v. Now find the edge leading to the assignment of value $\lambda_{v'}$ and so on. We get

$$v, v', v'', v''', \ldots, s.$$

It is not possible to have repeats in this list, because then G would have contain a cycle of negative length. We conclude that

$$s, \ldots, v''', v'', v', v$$

is the desired path. □

Theorem 1.6.10 *For a graph G with weights $w : E(G) \to \mathbb{R}$ and no negative cycles, Algorithm 1.6.5 will terminate in finitely many steps with $\lambda_v = d(s, v)$ for all $v \in V$.*

Proof. To prove termination, we first notice that by Lemma 1.6.9 any value λ_v in the algorithm is the length of some path in G. Because G has only finitely many paths, λ_v can attain only finitely many values. However, in each iteration, some λ_v value is lowered. Therefore the algorithm terminates.

When the algorithm has terminated, then $\lambda_v \geq d(s, v)$ because of the existence of the (s, v)-path constructed in Lemma 1.6.9 of length λ_v. Suppose for contradiction that $\lambda_v > d(s, v)$ for some v. Let $s = v_0, v_1, \ldots, v_k = v$ be a shortest path from s to v. Let i be the smallest i with $\lambda_{v_i} > d(s, v_i)$. Then consider the arc (v_{i-1}, v_i). We have

$$\lambda_{v_i} > d(s, v_i) = w_{i-1,i} + d(s, v_{i-1}) = w_{i-1,i} + \lambda_{v_{i-1}}.$$

The first equality follows from $v_0, \ldots, v_k$ being a shortest path and the second from i being minimal. But because the algorithm terminated, we cannot have

$$\lambda_{v_i} > w_{i-1,i} + \lambda_{v_{i-1}}$$

because this is the condition of the while loop. This is a contradiction. We conclude that $\lambda_v = d(s, v)$. □

Remark 1.6.11 Algorithm 1.6.5 can easily be modified so that it only performs a polynomial number of steps. In particular the behaviour in Exercise 1.6.8 will not appear. Simply order the edges $e_1, \ldots, e_m$, and always search for the (i, j) edge in the algorithm by cyclically going through this list, starting at the spot where the previous (i, j) edge was found. It can be proved that the number of times it is required to cycle through the list is at most $|V|$. Therefore the whole procedure takes time in the order of $O(|V| \cdot |E|)$.

By this remark we have solved the problem we set out to solve for the Hungarian Algorithm 1.5.3, namely to find an efficient method for finding w-maximal weighted augmenting paths from X to Y in G_M with weighting w_M. Only one run of the Bellman-Ford-Moore Algorithm is required, if we add additional vertices as in Figure 14.

1.6.3 Floyd's Algorithm

Even if it is not necessary for the Hungarian Algorithm 1.5.3, we now discuss the problem of finding all shortest distances in a graph with edge weights. That is we want to compute a matrix $D \in (\mathbb{R} \cup \{\infty\})^{V \times V}$ with $D_{u,v} = d(u, v)$ for all vertices $u, v \in V$. This matrix is called a distance matrix.

We have several possibilities:

- Run Dijkstra's Algorithm 1.6.1 n times. That will take time $O(|V|^3)$, but negative weights are not allowed.
- Run the Bellman-Ford-Moore Algorithm 1.6.5 n times. That will take time $O(|V|^2|E|)$ which is often as bad as $O(|V|^4)$ if there are many edges.
- Run Floyd's Algorithm 1.6.12 which we will now present. This will take time $O(|V|^3)$.

The idea in Floyd's Algorithm is to look for the pattern of Figure 16. But unlike Warshall's Algorithm 1.6.4 we this time check if the previously known distance between j and k is larger than the shortest known distance between j and i and the shortest known distance between i and k. We keep track of the known short distances in the matrix Λ and update it in a way similar to the updates of the adjacency matrix of Warshall's Algorithm. To make things simpler, we unlike Warshall's Algorithm

Algorithm 1.6.12 *Floyd's Algorithm*
Input: *A directed graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}$. The graph G should contain no cycles of negative length.*
Output: *The distance matrix D of G.*

- *Let* $\Lambda_{u,v} := \begin{cases} 0 & \text{for } (u, v) \text{ with } u = v \\ w(u, v) & \text{for } (u, v) \in E \text{ with } u \neq v \\ \infty & \text{otherwise} \end{cases}$
- *For* $i = 1, 2, \ldots, |V|$
  - *For* $j = 1, 2, \ldots, |V|$
    - *For* $k = 1, 2, \ldots, |V|$
      - *Let* $\Lambda_{jk} := \min(\Lambda_{jk}, \Lambda_{ji} + \Lambda_{ik})$.
- *Return* $D := \Lambda$.

In the algorithm we have ordered the vertices V so that they can be used to index columns and rows of the matrix Λ. Moreover, for the initialisation of $\Lambda_{u,v}$, if repeated (u, v) arcs exist, $\Lambda_{u,v}$ should be the smallest w-value of those. The ideas of the following proof are very similar to those of the proof of Warshall's Algorithm 1.6.4.

Exercise 1.6.13 If we have run Algorithm 1.6.12, how could we find a shortest path given start and end vertices? Can this be done easier if we modify the algorithm? How?

Exercise 1.6.14 Are Floyd's and Warshall's algorithm the same for graphs where all weights are 0?

Proof. Since $|V|$ is constant, the algorithm clearly terminates.

To prove correctness, let Λ^i be the value of Λ after the ith iteration of the outer loop. Clearly we have $\Lambda^0 \geq \Lambda^1 \geq \cdots \geq \Lambda^{|V|} \geq D$ entry-wise, where the last inequality follows because we only assign lengths to Λ if a walk of that length indeed exists between the vertices in question.

To prove that $\Lambda^{|V|} \leq D$, we prove the statement $S_0, \ldots, S_{|V|}$ by induction:

S_i: For $(s,t) \in V \times V$, $\Lambda^i_{s,t}$ is $\leq$ the w-length of a shortest (s,t)-walk in G with all vertices except s and t being among the i first vertices.

In the base case $i = 0$, the statement S_i is that if there is a (s,t)-walk in G involving no other vertices than s and t then it has length $\geq \Lambda_{s,t}$. This is true either because $s = t$ and $\Lambda_{s,t} = 0$ or because the walk has $s \neq t$ and therefore at least of length $\Lambda^0_{s,t}$. (We have used that G has no cycles of negative length!)

We now prove $S_{i-1} \Rightarrow S_i$. Suppose S_{i-1} is true. Notice that it suffices to prove for any (s,t)-walk P only involving $s, t, 1, \ldots, i$ that

$$\Lambda^i_{s,t} \leq w(P).$$

- If vertex i is not involved in P then S_{i-1} implies $w(P) \geq \Lambda^{i-1}_{s,t} \geq \Lambda^i_{s,t}$ as desired.
- If i is involved in P, then consider an (s,i)-subwalk Q_1 of P and an (i,t)-subwalk Q_2 of P. We choose Q_1 and Q_2 to be such that they have a minimal number of edges. Because G has no cycles of negative weight

  $$w(P) \geq w(Q_1) + w(Q_2) \geq \Lambda^{i-1}_{s,i} + \Lambda^{i-1}_{i,t} \geq \Lambda^i_{s,t}.$$

  The second inequality follows from S_{i-1} because the interior vertices of Q_1 and Q_2 all have index at most $i-1$. The last inequality follows from the way that $\Lambda^i_{s,t}$ gets its value in the assignment in the algorithm.

By induction S_i is true for any i. In particular, when $i = |V(G)|$ it gives $\Lambda^{|V|} \leq D$. □

1.7 Maximal weight matching in general graphs

We return to the situation of non-directed graphs and the problem of finding maximal weight matchings. As motivation for the non-bipartite case we consider the Chinese postman problem. Both this motivating example and and the proof of Theorem 1.7.6 below require that we allow graphs to have parallel edges. (Note that this does not cause problems even when applying Floyd's Algorithm 1.6.12 in Algorithm 1.7.2 below, as Floyd's Algorithm can easily be modified to deal with the parallel-arc situation.) Moreover, we will use the notation $G + H$ for the graph with vertex set $V(G) \cup V(H)$ and edge sets being the union edges in G and H, but with edges repeated if they appear in both $E(G)$ and $E(H)$. In particular $|G + H| = |G| + |H|$.

1.7.1 The Chinese postman problem

The following problem was studied by the Chinese mathematician Kwan:

We consider the street map of a town. The streets have lengths and the map may be represented as a graph $G = (V, E)$ with a weight function $w : E \to \mathbb{R}_{\geq 0}$. The post office is located at one of the vertices v in G. At the post office there is a single postman working. He needs to deliver mail along each street on the map (or edge in the graph). The streets are narrow and it is not necessarily necessary to walk a long a street more than once. The problem is to find a closed walk P starting and ending at the post office vertex v so that all edges have been covered by P at least once and $w(P)$ is minimal. We will call such walk an optimal postman tour. (A *tour* is the name for a closed walk covering all edges at least once.)[12]

We will assume that the street map graph is connected. Recall (from Graph Theory 1) that an *Eulerian tour* in a graph is a tour using each edge of the graph exactly once. A graph is *Eulerian* if it has an Eulerian tour. This is the case if and only if the graph is connected and all vertices have even degree. In this case an Eulerian tour can then be found using Fleury's Algorithm from Graph Theory 1. Because we have assumed that all edges have non-negative length, the Euler tour would be an optimal solution to the Chinese postman problem. However, not every graph is Eulerian.

If G is not Eulerian, let U be the set of vertices with odd degree. These are called *odd* vertices. A possible approach now is to add edges to G (introducing parallel edges) so that G becomes Eulerian and then find an Eulerian tour.

Lemma 1.7.1 *Given a connected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}_{\geq 0}$ there exists a set of walks $P_1, \ldots, P_k$ in G with the ends being odd vertices in G (with each odd vertex being an end of one walk) such that adding the walks to G by duplicating edges, the new graph $G + P_1 + \cdots + P_k$ is Eulerian and each of its Euler tours (starting at v) is an optimal postman tour in G.*

Proof. Let W be an optimal postman tour in G. Then W is an Eulerian tour in a graph G' where edges have been repeated in G so that W uses each edge in G' exactly once. Consider now the graph G'' being G', but with the original edges of G removed. That is, G'' contains the edges that were added to G to make it even. The graphs G'' and G have the same set of odd vertices U. It is not difficult to decompose G'' into $|U|/2$ walks $P_1, \ldots, P_{|U|/2}$ and an even, possibly empty, subgraph H: $G'' = P_1 + \cdots + P_{|U|/2} + H$. (For example, find P_1 by starting a walk from an odd vertex until another odd vertex is reached. If a vertex was visited more than once, then remove edges and vertices from P_1 to make it a true path between two odd vertices of G''. To find P_2 consider G with $E(P_1)$ removed, and so on. Eventually we are left with a subgraph with only even vertices.) We now have

$$w(G') - w(G) = w(G'') = w(P_1) + \cdots + w(P_{|U|/2}) + w(H) \geq w(P_1) + \cdots + w(P_{|U|/2})$$

[12] Please be aware of the difference between a path and a walk when reading this. See Bondy and Murty [1] for definitions.

because the even subgraph H must have non-negative weight. This implies $w(P_1) + \cdots + w(P_{|U|/2}) + w(G) \leq w(G') = w(W)$. Because G with the paths $P_1, \ldots, P_{|U|/2}$ added is Eulerian and has weight at most $w(W)$ it has an Euler tour being an optimal postman tour in G. □

We notice that each P_i mentioned in the lemma must be a w-shortest walk between the ends of P_i. (If P_i' was w-shorter, then the Eulerian graph $G + P_1 + \cdots + P_{i-1} + P_i' + P_{i+1} + \cdots + P_k$ would have smaller w-weight than the optimal tour.) Therefore, it is a consequence of the lemma that it suffices to look for walks with their set of ends being U and total w-length minimal. These walks can be obtained via the computation of a max w-weight matching in a general graph, leading to the following algorithm.

Algorithm 1.7.2
Input: *A graph G with a weight function $w : E(G) \to \mathbb{R}_{\geq 0}$*
Output: *An optimal postman tour P in G.*

- *Find the vertices U of odd degree in G*
- *Use Floyd's Algorithm 1.6.12 to find a w-shortest (x, y)-path $P_{x,y}$ for all $x, y \in V$ and $x \neq y$.*
- *Define the complete graph K_U with vertex set U and weighting w' with $w'(x, y) := -w(P_{x,y})$.*
- *Find a max w'-weight perfect matching M in K_U (using the ideas of Section 1.7.2).*
- *For each $(x, y) \in M$ add the path $P_{x,y}$ to G by duplicating edges.*
- *Run Fleury's Algorithm on the new Eulerian graph G to obtain an Eulerian path P.*
- *Return P (considered as a tour in the original G).*

Example 1.7.3 Consider the graph on the top left in Figure 20. We find an optimal postman tour by applying the algorithm. Notice that the three $P_{x,y}$ paths that need to be added consist of $1, 1$ and 2 edges respectively.

1.7.2 Edmonds' perfect matching polytope theorem

We generalise the perfect matching polytope discussed in Exercise 1.1.3 for bipartite graphs to general graphs.

Definition 1.7.4 Given a graph $G = (V, E)$ and a subset $M \subseteq E$, we define the *characteristic vector* (or *incidence vector*) $\chi_M \in \mathbb{R}^E$ with coordinates $(\chi_M)_e = 1$ if $e \in M$ and $(\chi_M)_e = 0$ if $e \notin M$. The *perfect matching polytope* $PMP(G)$ of a graph G is defined as

$$PMP(G) := \text{conv}(\{\chi_M : M \text{ is a perfect matching in } G\}).$$

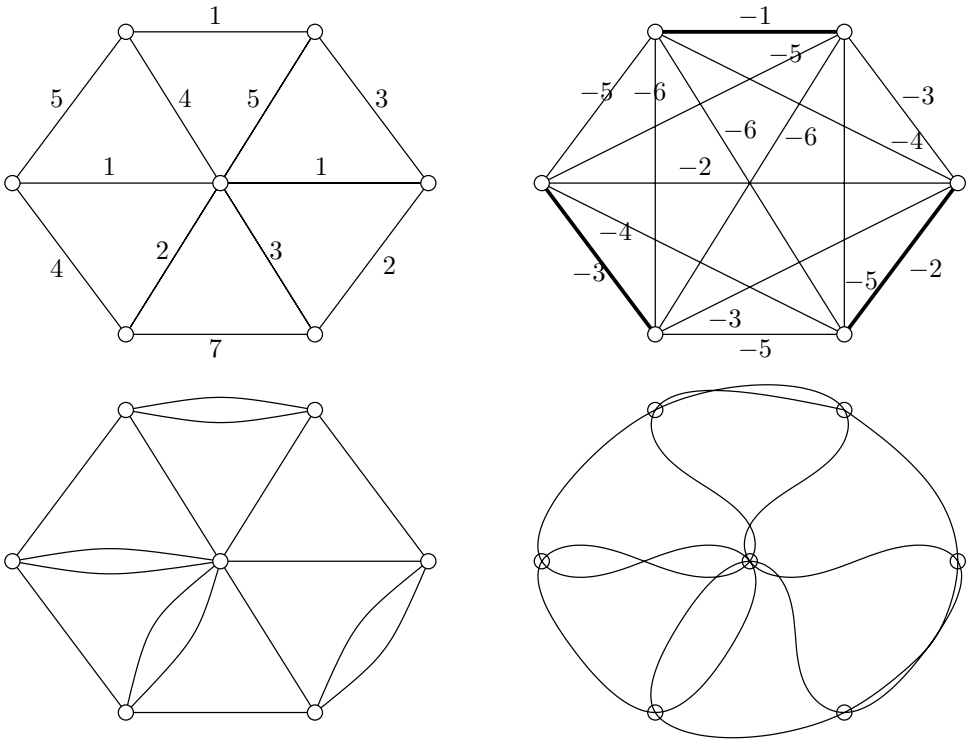

Figure 20: The graph G in Example 1.7.3 with its weighting w. The graph K_U with its weighting w' and a max weight perfect matching. The graph G after adding three paths. An Eulerian tour in the new G graph - that is, an optimal postman tour in the old G graph.

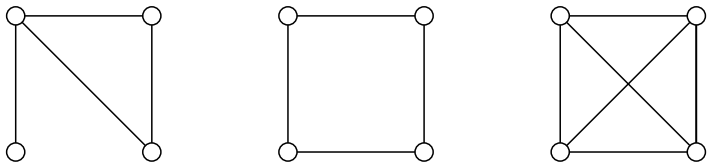

Figure 21: The three graphs in Exercise 1.7.5.

We notice that if $|V(G)|$ is odd, then G has no perfect matching and $PMP(G) = \emptyset$.

Exercise 1.7.5 Find $PMP(G)$ for each of the graphs in Figure 21.

Recall that for any subset $U \subseteq V$, we define the *edge cut* $\partial(U)$ as the set of edges in G with one end in U and the other in $V \setminus U$. We use the weight notation of the previous sections and in particular write $x(\partial(U)) := \sum_{e \in \partial(U)} x_e$ for $x \in \mathbb{R}^E$.

In the following theorem (and in particular in its proof) we allow the graph to have parallel edges. A similar proof can be found in [3].

Theorem 1.7.6 (Edmonds' perfect matching polytope theorem) *For a loop-free graph G the perfect matching polytope is also described by the inequalities:*

1. $x_e \geq 0$ for all $e \in E$
2. $x(\partial(\{v\})) = 1$ for all $v \in V$
3. $x(\partial(W)) \geq 1$ for all $W \subseteq V$ *with* $|W|$ *odd*

where $x \in \mathbb{R}^E$.

Proof. Let $Q \subseteq \mathbb{R}^E$ be the solution set to the inequalities above. We argue that $Q = PMP(G)$.

To prove $Q \supseteq PMP(G)$, we observe that for any perfect matching M the characteristic vector χ_M satisfies equations (1) trivially, equations (2) because M is a perfect matching, and equations (3) because at least one vertex in W is not matched with vertices from W in M.

To prove $Q \subseteq PMP(G)$, first consider the case where G is a cycle or a disjoint union of cycles. We prove this case in Exercise 1.7.9.

For general graphs, suppose that there were graphs where $Q \subseteq PMP(G)$ was not the case, and furthermore suppose that we have a counter example G with $|V| + |E|$ as small as possible where it is not the case. We know that both $PMP(G)$ and Q are polytopes (the second set is bounded). Because $Q \not\subseteq PMP(G)$ there must exist a vertex x of Q which is not in $PMP(G)$.

If for some $e \in E$ we have $x_e = 0$, then the smaller graph $G \backslash e$ would also be a counter example (Exercise 1.7.10). Similarly if $x_e = 1$ the smaller graph G with e and its ends removed would also be a counter example (Exercise 1.7.11). The

counter example G has no isolated vertex v, since in that case $PMP(G) = \emptyset$ and also the type two inequality for v cannot be satisfied. Therefore $Q = PMP(G)$ and G would not be a counter example. The graph has no vertex v of degree 1, because then the incident edge e would have $x_e = 1$. Therefore all vertices have degree ≥ 2. Not all degrees can be 2 because G is not a union of cycles. Therefore $|V| < |E|$.

The vertex x is the intersection of $|E|$ hyperplanes in $\mathbb{R}^E$ obtained by changing the inequalities above to equations. They cannot be of type 1, and only $|V|$ of them can be of type 2. Therefore one of the equations must be of type 3. Hence there exists $W \subseteq V$ with $|W|$ odd so that $x(\partial(W)) = 1$. If $|W| = 1$ then this equation would be of type 2. Therefore we assume $|W| \geq 3$.

Let G' be the graph obtained by contracting W to a single vertex u'. Let x' the vector induced by x on the smaller edge set. Edges disappear when they have both ends in W. However, parallel edges may appear in the contraction. The vector x' is a coordinate projection of x. This x' satisfies inequalities 1-3 for the smaller graph G' (Exercise 1.7.12). Similarly we define G'' and x'' by contracting $V \setminus W$. The projection x'' of x satisfies inequalities 1-3 for the graph G''.

Both G' and G'' are smaller than G and therefore we can write

$$x' = \sum_{M'} \lambda_{M'} \chi_{M'} \text{ and } x'' = \sum_{M''} \mu_{M''} \chi_{M''}$$

where the first sum runs over all M' being matchings in G', the second sum runs over all M'' being matchings in G'' and $\sum_{M'} \lambda_{M'} = 1 = \sum_{M''} \mu_{M''}$. Because all data is rational we can assume (essentially by Cramer's rule) that $\lambda_{M'}, \mu_{M''} \in \mathbb{Q}$ or rather that there exists $k \in \mathbb{N} \setminus \{0\}$ such that

$$kx' = \sum_{M'_i} \chi_{M'_i} \text{ and } kx'' = \sum_{M''_i} \chi_{M''_i}$$

where the sum run over sets of matchings $M'_1, \ldots, M'_r$ and $M''_1, \ldots, M''_s$ with possible repeats. Let $e \in \partial(W)$. The coordinate $(kx')_e$ is the number of times that e appears in an M'_i. Because x' and x'' are projections of x we also have $(kx)_e = (kx')_e = (kx'')_e$. Therefore the number of times that e appears in $M'_1, \ldots, M'_r$ is the number of times that e appears in $M''_1, \ldots, M''_s$. Since this holds for each e we may assume after reordering (because M'_i and M''_i contain exactly one edge from $\partial(W)$ each) that for all i, M'_i and M''_i contain the same edge from $\partial(W)$. Therefore $M'_i \cup M''_i$ is a matching in G. We conclude that $kx = \sum_i \chi_{M''_i \cup M'_i}$ and therefore x is a convex combination of the characteristic vectors of the perfect matchings $M'_1 \cup M''_1, \ldots, M'_r \cup M''_r$. A contradiction. □

Example 1.7.7 Let's verify the theorem for the complete graph K_4 using the online polytope calculator "Polymake"[13]: http://shell.polymake.org . In this case inequalities of type 3 are redundant (why?). The polytope lives in $\mathbb{R}^6$. The

[13]The polymake system conveniently ties many pieces of polytope software together and allows the user to use them in a uniform way. The main developers are Gawrilow and Joswig, but through the included software the system has contributions from over a hundred people.

inequalities of type 1 are encoded by a 6×7 matrix A, where the first column i treated specially. Similarly, the inequalities of type 2 are encoded by a 4×7 matrix B. We hand this H-description to Polymake and ask for the vertices:

```
polytope > $A=new Matrix<Rational>([[0,1,0,0,0,0,0],[0,0,1,0,0,0,0],
[0,0,0,1,0,0,0],[0,0,0,0,1,0,0],[0,0,0,0,0,1,0],[0,0,0,0,0,0,1]]);

polytope > $B=new Matrix<Rational>([[-1,1,1,1,0,0,0],
[-1,1,0,0,1,1,0],[-1,0,1,0,1,0,1],[-1,0,0,1,0,1,1]]);

polytope > $p=new Polytope<Rational>(INEQUALITIES=>$A,EQUATIONS=>$B);

polytope > print $p->VERTICES;
1 0 0 1 1 0 0
1 0 1 0 0 1 0
1 1 0 0 0 0 1
```

Back we get the vertices $(0,0,1,1,0,0), (0,1,0,0,1,0), (1,0,0,0,0,1)$ which are also the characteristic vectors of the three perfect matchings in K_4.

Exercise 1.7.8 Give an example of a graph G with a loop where the statement of Theorem 1.7.6 is false.

Exercise 1.7.9 Fill the first gap in the proof of Theorem 1.7.6 by proving the theorem in the case where

- G is a cycle.
- G is a disjoint union of cycles.

Exercise 1.7.10 Fill the second gap in the proof of Theorem 1.7.6.

Exercise 1.7.11 Fill the third gap in the proof of Theorem 1.7.6.

Exercise 1.7.12 Fill the fourth gap in the proof of Theorem 1.7.6.

1.7.3 A separation oracle

We discuss the following problem. Given a graph $G = (V, E)$ and a point $x \in \mathbb{R}^E$, how do we decide if $x \in PMP(G)$ and moreover, if $x \notin PMP(G)$, how do we find a vector $w \in \mathbb{R}^E$ such that $\omega \cdot x > \omega \cdot y$ for all $y \in PMP(G)$? That is, how do we find a separating hyperplane between x and $PMP(G)$? A method for answering such a question is called a *separation oracle*. If a quick separation oracle exists, then the so called *ellipsoid method* will give a (theoretically) quick method for maximising linear functions over $PMP(G)$.

It is easy to check the two first sets of linear inequalities/equations of Theorem 1.7.6 by plugging in the coordinates of x. Suppose they are all satisfied. The third set, however, contains exponentially many inequalities (expressed as a function of $|V|$). Rather than substituting, we wish to compute

$$\min_{W \subseteq V \text{ with } |W| \text{ odd}}(x(\partial(W))) \tag{4}$$

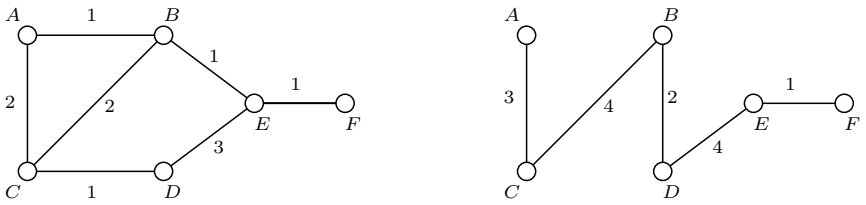

Figure 22: The graph of Example 1.7.14 with its weighting w. The Gomory-Hu tree with its weighting.

and a subset W where the value is attained. If the value is ≥ 1 then all exponentially many inequalities of Theorem 1.7.6 are satisfied. If not, then at least one inequality (for example the one indexed by W) is not satisfied.

To compute the minimum efficiently we need to introduce so called *Gomory-Hu trees*. In the following we consider $G = (V, E)$ with a weight (or capacity) function $w : E \to \mathbb{R}_{\geq 0}$. For $s, t \in V$ with $s \neq t$, an (s, t)-cut is an edge cut of the form $\partial(W)$ with $W \subseteq V$ and $s \in W \not\ni t$.

Definition 1.7.13 A *Gomory-Hu tree* of G is a tree T with $V(T) = V(G)$ and with weights $w' : E(T) \to \mathbb{R}_{\geq 0}$ such that the following property holds:

- For all $s \neq t$ the edge e on the path sTt with $w'(e)$ minimal defines a cut $\partial(W)$ by letting W be the vertices of one of the connected components of $T \setminus e$. Moreover, this cut is a w-minimal (s, t)-cut and has weight $w'(e)$.

Example 1.7.14 Consider the graph G of Figure 22 with weighting (capacities) w. The weights of all w-minimal (s, t)-cuts in G are shown in the following table.

	A	B	C	D	E	F
A		3	3	2	2	1
B	3		4	2	2	1
C	3	4		2	2	1
D	2	2	2		4	1
E	2	2	2	4		1
F	1	1	1	1	1	

For example, the smallest w-weight of an (A, C)-cut is 3. All the values in the table are encoded in the Gomory-Hu tree shown on the right in the figure. For example, the unique path in this tree from A to C contains two edges. The smallest weight edge of these two has weight 3.

Exercise 1.7.15 Choose some graph with a weighting. Find its Gomory-Hu tree.

A Gomory-Hu tree of a graph can be computed via the Gomory-Hu Algorithm, which we will not present. The algorithm runs in polynomial time.

Any non-trivial edge cut is an (s,t)-cut for some vertices s and t. Therefore, we can compute

$$\min_{W \subseteq V \text{ with } |W| \notin \{0,|V|\}} (x(\partial(W))) \tag{5}$$

simply by finding an edge in T with minimal w' weight. However, this is not the minimum we want to compute in (4), since we want to restrict ourselves to *odd* cuts $\partial(W)$. (Meaning $|W|$ must be odd).

We explain how Gomory-Hu trees can also be used for this computation.

From now on we now assume that $|V(T)|$ is even. An edge of T is called *odd* if, by removing it from T, the set $V(T)$ splits into two connected components of odd size.

Algorithm 1.7.16 (Padberg-Rao)
Input: *An graph $G = (V, E)$ with $|V| > 0$ and even and a weight function $w : E \to \mathbb{R}_{\geq 0}$.*
Output: *An odd set $W \subset V$ such that $w(\partial(W))$ is minimal.*

- *Find the Gomory-Hu tree T with weights w'.*
- *Find an odd edge e such that $w'(e)$ is minimal and let W be the vertices of one of the components of $T \setminus e$.*

Proof. We wish to prove that W defines a cut with weight at least as small as that of any other cut of an odd sized set. Let $W' \subseteq V$ with $|W'|$ odd, defining a cut $\partial(W')$. The vertex-induced subgraph[14] $T[W']$ is a forest[15] with an odd number of vertices. Therefore, one of the components in $T[W']$ must be a tree T' with an odd number of vertices. Because $W' \neq V$ there must be an edge $e' = (s,t) \in E(T)$ with $s \in V(T')$ and $t \in V \setminus W'$. In particular $e' \in \partial(W')$. We now claim

$$w(\partial(W')) \geq w'(e') \geq w'(e) = w(\partial(W)).$$

To prove this, first observe that T with weights w' being a Gomory-Hu tree implies that $w'(e')$ is the smallest weight of an (s,t)-cut. On the other hand, $\partial(W')$ is such an (s,t)-cut. This proves the first inequality. The second follows from the algorithm's choice of e. The equality follows because W was chosen as the vertices of one component of $T \setminus e$ and T with weights w' is a Gomory-Hu tree. □

The algorithm is an efficient method for checking the exponentially many type 3 inequalities of Edmonds' perfect matching polytope theorem (Theorem 1.7.6). To check whether a vector $x \in \mathbb{R}^E$ satisfies all type 3 inequalities, simply compute $\min_{W \subseteq V, |W| \text{ odd}} x(\partial(W))$. If this number is smaller than 1 then one of the inequalities is not satisfied. Which one is determined by the W found by Algorithm 1.7.16. This completes the description of the separation oracle.

[14] See Appendix A.
[15] A graph is called a forest if all its components are trees.

1.7.4 Some ideas from the ellipsoid method

Suppose P is a polytope (bounded polyhedron) which we do not know. Suppose we also know that

- P is contained in some big ball $B \subseteq \mathbb{R}^n$
- P is either full-dimensional or empty,
- if P is non-empty, then its volume is at least 1,
- we have a separation oracle, which means that we can ask for any point $x \in \mathbb{R}^n$ if $x \in P$ and if it is not, then we get a hyperplane H with x on one side $x \in H^-$ and P on the other side $P \subseteq H^+$.

Initially we know $P \subseteq B$. By using the oracle wisely, we will either guess a point in P or we will produce smaller and smaller sets say $B_1, B_2, \ldots$ containing P by letting $B_i = B_{i-1} \cap H_{i-1}^+$ as we go along. By clever choices of the point x, we can make the volume of B_i arbitrary small as i increases. If the volume of B_i is less than 1, then P must be empty. Therefore, this gives a method for deciding if P is empty.

One problem with the method is that at each step the set B_i is described as the intersection of B with i halfspace. Therefore the representation gets more and more complicated as i increases. One method that avoids this is the ellipsoid method where the set B_i is represented by a slightly larger ellipsoid E_i. See Figure 23 for the first few steps of a run of the ellipsoid method. The rate at which the volumes of these ellipsoids decrease allowed Khachiyan to prove that the ellipsoid method can be used to solve Linear Programming problems in polynomially many iterations (in the size of the LP problem).

Our LP problem is exponential in size (as a function of $|V|$) but we still have a separation oracle. Therefore the ellipsoid method can be applied. In fact the method can be used to maximise a linear function over polyhedron for example the perfect matching polytope. We have skipped a lot of things in this presentation of the ellipsoid method. For example our matching polytope is not full-dimensional, and if the ellipsoid method did find an $x \in P$, this x might not be a vertex of the matching polytope (since it did not have $0, 1$-coordinates), and we would not now how to recover the matching.

For the purpose of this class however, we consider the problem solved, when we know an efficient separation oracle.

Remark 1.7.17 An important remark on the ellipsoid method is that it is mainly of theoretical interest and does not perform that well in practise. For example for Linear Programming where the constraints are written as a matrix, interior point methods exists which work well both in theory and practise, and one would never use the ellipsoid method in this setting.

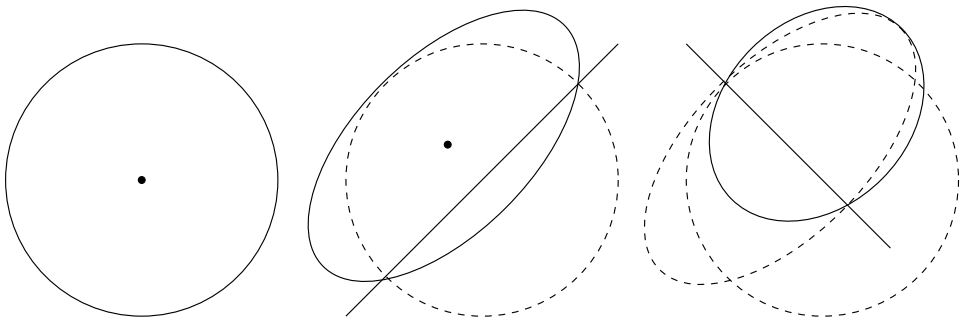

Figure 23: Some iterations of the ellipsoid method. The polytope P is hidden somewhere in the intersection of all the ellipsoids. Using the separation oracle, a hyperplane is produced, and we now know that P must be hidden in the smaller ellipsoid in the middle picture. Using the center of the ellipsoid as x we get a new separating hyperplane, and we conclude that P must be contained in the even smaller ellipsoid in the picture to the right. The process is repeated until a point in P is guessed, or the ellipsoids are so small that the conclusion is that P is empty.

2 Matroids

Before talking about matroids we will first do the following two exercises as a warm up and also briefly study planar graphs.

Exercise 2.0.1 Consider the matrix

$$\begin{pmatrix} 1 & 1 & 0 & 2 & 2 \\ 3 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Let's assign the names a, b, c, d and e to the columns of the matrix. The columns a and b are linearly independent. Therefore we call $\{a, b\}$ an *independent* set of columns. Find all independent sets of columns of the matrix. Among these find all the maximal independent sets. What are their sizes? What are the minimal non-independent sets? Do they have the same size?

Exercise 2.0.2 Consider the complete graph K_4 with 4 vertices and 6 edges. Let's call a (spanning) subgraph of K_4 *independent* if it does not contain a cycle. How many independent (spanning) subgraphs of K_4 are there? What are the maximal independent subgraphs? What kind of graphs are they? How many edges do they contain? What are the minimal non-independent subgraphs? Do they have the same number of edges?

2.1 Planar graphs

Definition 2.1.1 A graph G is *planar* if we can draw it in the plane $\mathbb{R}^2$ such that edges only overlap at vertices.

Example 2.1.2 The complete graph K_4 is planar. See Figure 24.

A drawing like those in Figure 24 is called an embedding of the graph. A graph together with an embedding is called a *plane graph*.

Remark 2.1.3 It is possible to prove that whether we require edges to be drawn as straight lines or as continuous paths in the plane does affect the notion of planar graphs as long as graphs are assumed to have no parallel edges. However, some of the edges will be parallel in our examples. Therefore we allow edges to be drawn as nonstraight lines.

Proposition 2.1.4 *The complete graph K_5 is not a planar graph.*

Proof. Suppose we had an embedding. Then we would have a closed path C_0 in the plane going through v_1, v_2, v_3, v_1. There now are two different situations, both shown in Figure 25.

In the first situation (left) v_4 is on the inside of C_0. We observe that v_5 must also be on the inside of C_0 since v_4 and v_5 are connected by an edge not intersecting C_0. On the inside of C_0 there are three regions. However, it is impossible for v_5 to be in any of these regions as it then could not be connected to the outside vertex by an edge without intersecting C_0.

Now we consider the case where v_4 is outside. Without loss of generality the situation is as drawn on the right in Figure 25. We now have the closed paths $C_1 = v_2v_3v_4v_2$, $C_2 = v_3v_1v_4v_3$, $C_3 = v_1v_2v_4v_1$. Because v_1 is outside the closed path C_1 and v_5 is connected to v_1 by a path, v_5 is outside C_1. Similarly we get that v_2 being inside C_2 implies v_5 being inside C_2 and v_3 being outside C_3 implies v_5 being outside C_3. In the picture, the only possibility for v_5 is that it is inside C_0. This contradicts v_4 being outside C_0 and v_4 and v_5 being connected by an edge. Hence K_5 has no embedding in the plane. □

We have used the following theorem (maybe without noticing):

Theorem 2.1.5 (Jordan's Curve Theorem) *Let $S^1 \subseteq \mathbb{R}^2$ be the unit circle and $f : S^1 \to \mathbb{R}^2$ an injective continuous function and let $C = f(S^1)$ be its image. Then $\mathbb{R}^2 \setminus C$ has exactly two connected components. One is bounded and the other is unbounded. The curve C is the boundary of each component.*

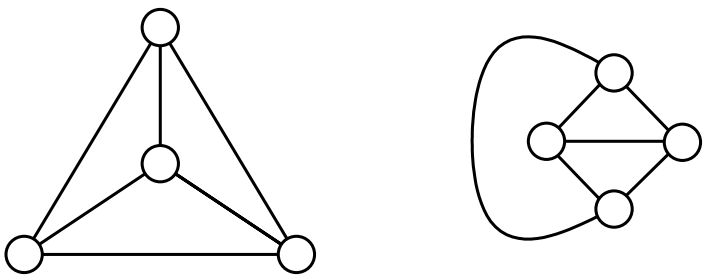

Figure 24: Two drawings of the graph K_4 in the plane.

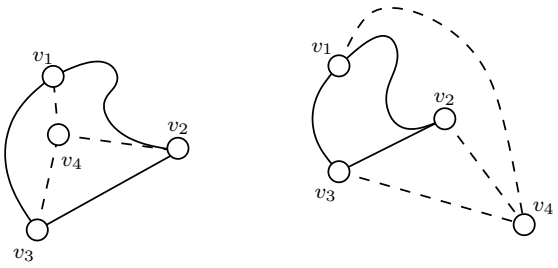

Figure 25: The two situations in the proof of Proposition 2.1.4.

The statement of the theorem is intuitively obvious. However, it is difficult to give a formal proof of the theorem. Such a proof is given in the Topology classes (for the pure mathematicians) - see also Exercise 4.5.8.

We used Jordan's Curve Theorem in the proof of Proposition 2.1.4. In particular we talked about a closed path in the plane having an "inside" and "outside", refering to the bounded and unbounded component.

Besides drawing graphs in the plane, a possibility is to draw graphs on the unit sphere. However, the graphs which can be embedded in the plane are exactly the graphs which can be embedded in the sphere. This is [1, Theorem 10.4]. The idea is to *stereographically* project the sphere to the plane.

Exercise 2.1.6 Can any graph G be embedded in $\mathbb{R}^3$? Here are two strategies/hints for solving the exercise:

- If four points $p_1, \ldots, p_4$ are uniformly and independently distributed over the cube $[0,1]^3$, what is the probability of straight line segments (p_1, p_2) and (p_3, p_4) intersecting?
- [1, Exercise 10.1.12] suggests to embed the vertices of the graph as different points on the curve

  $$\{(t, t^2, t^3) : t \in \mathbb{R}\}$$

  and make each edge a straight line. Why would that work?

2.2 Duality of plane graphs

We now consider plane graphs. That is, we consider graphs which are embedded in the plane. We let G also denote the embedding of a graph G. The set $\mathbb{R}^2 \setminus G$ is then a disjoint union of connected components. We call these components faces of G.

Example 2.2.1 The plane graph on the left in Figure 26 has four faces.

Definition 2.2.2 Let G be a plane graph. Its *dual* graph G^* is defined to be the graph which has a vertex for each face of G and an edge e^* for each edge e in G such that e^* connects the vertices for the faces that e separates.

Example 2.2.3 The dual graph of the plane graph of Example 2.2.1 is shown to the right of Figure 26.

Remark 2.2.4 Since we do not provide a specific embedding of G^* (we do not give actual coordinates of vertices and edges), we should really think of G^* as an equivalence class of plane graphs, where plane graphs are equivalent if we can continuously deform them into each other without letting edges overlap on the way. In Figure 26 we actually have a choice of letting the left most edge in G^* connect the lower end to the upper as indicated or letting the edge go from the lower end to the upper end around the graph counter clockwise. The two possibilities of dual graphs are not equivalent if considered as embeddings in the plane. However, if the graphs are drawn on the sphere, then they can be continuously deformed into each other, and we should regard them as equivalent.

One can prove that if G is a connected plane graph, then $(G^*)^* = G$ (in the sense of equivalence described above). It is easy to observe in many examples. (Why does the graph have to be connected?)

It is extremely important that the graph is plane when talking about its dual as the following example shows.

Example 2.2.5 (from wikipedia) The underlying graph of the two plane graphs of Figure 27 are isomorphic. However, their dual graphs are not. (Left as an exercise).

2.3 Deletion and contraction in dual graphs

Recall that when we delete an edge from a graph we leave the remaining vertices and edges untouched. The number of edges decreases by one, while (for a plane graph) the number of faces decreases (unless the same face is on both sides of the edge).

When we contract an edge on the other hand, two vertices get identified. The number of edges decreases by one and so does the number of vertices, while (for plane graphs) the number of faces stays the same (unless the contracted edge is a loop).

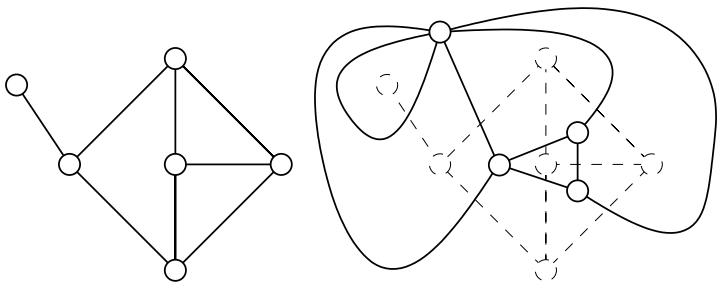

Figure 26: The plane graph of Example 2.2.1 and its dual (see 2.2.3

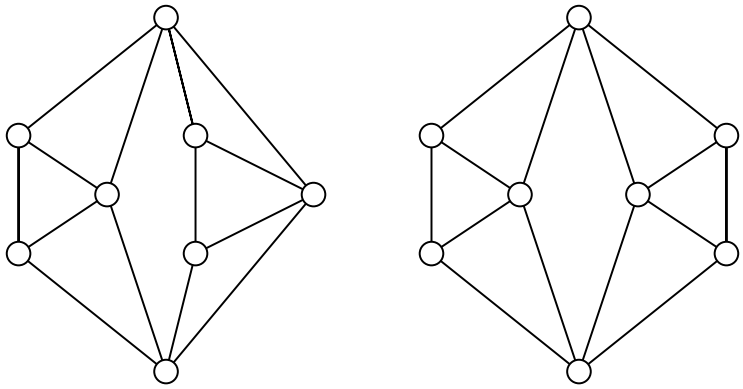

Figure 27: The two plane graphs are isomorphic, but their duals are not. See Example 2.2.5.

Example 2.3.1 In Figure 28 we consider a plane graph G. First deleting an edge e and then taking its dual is the same as taking the dual G^* and contracting the edge e^* in it.

2.4 Matroids

A matroid is a mathematical object which captures many properties of a graph. We can define deletions and contractions for a matroid and every matroid has a dual matroid. In this sense a matroid fixes one of the problems we have with graphs. However, not every matroid comes from a graph.

Matroids can be defined by specifying a set of *axioms* that the matroid must satisfy. There are several equivalent ways of defining matroids (Definition 2.4.2, Theorem 2.6.1, Theorem 2.7.4, Theorem 2.8.4). We start by defining them in terms of independent sets.

2.4.1 Independent sets

Imagine that we have a matrix A with n columns indexed by a set S. We call a subset of S *independent* if the columns of A indexed by the subset are linearly independent. Observe that the following three properties hold:

- The empty set of columns $\emptyset$ is independent.
- If $I \subseteq S$ is independent, then so is any $J \subseteq I$.
- If A, B are independent subsets of columns, with $|B| = |A| + 1$ then there exists $b \in B \setminus A$ such that $A \cup \{b\}$ is independent.

Usually the question of whether $\emptyset$ is independent is ignored in linear algebra. However, if the definition of linear independence is read carefully then $\emptyset$ is indeed independent. The second claim follows easily from the definition of linear independence while the last needs a tiny bit of work.

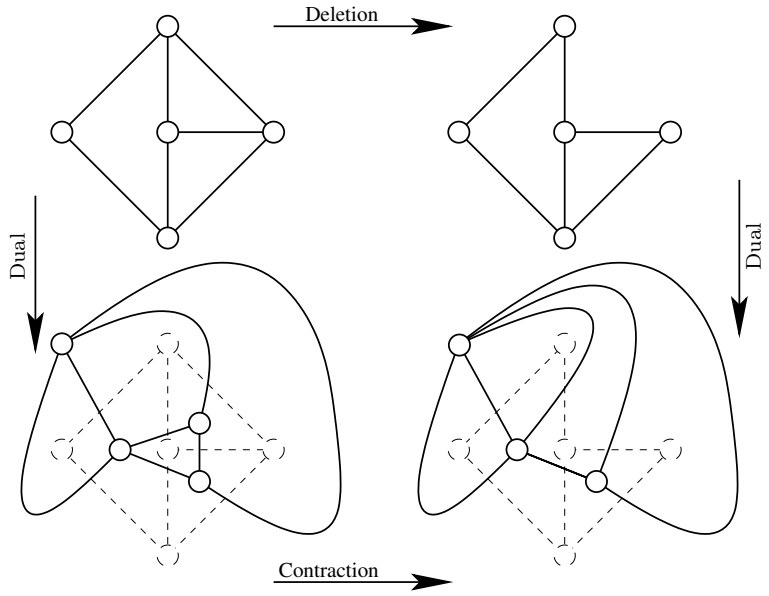

Figure 28: Doing a deletion on a graph corresponds to doing a contraction on the dual. See Example 2.3.1.

Exercise 2.4.1 What is the definition of a set of vectors being dependent? Why is $\emptyset$ independent? Prove that the other two properties above hold.

A matroid captures the essence of independence:

Definition 2.4.2 Let S be a finite set and $\mathcal{I}$ a set of subsets of S. The pair $M = (S, \mathcal{I})$ is called a *matroid* with *ground set* S if the following hold:

I1: $\emptyset \in \mathcal{I}$.

I2: If $A \in \mathcal{I}$ and $B \subseteq A$ then $B \in \mathcal{I}$.

I3: If $A, B \in \mathcal{I}$ and $|B| = |A| + 1$ then there exists $b \in B \setminus A$ such that $A \cup \{b\} \in \mathcal{I}$.

The set in $\mathcal{I}$ is called the *independent* sets of the matroid M. A subset of S which is not independent is called dependent.

Definition 2.4.3 Let A be a matrix with real entries and columns indexed by a set S. The *vector* matroid of A is the pair $(S, \mathcal{I})$ where

$$\mathcal{I} := \{B \subseteq S : \text{the columns of } A \text{ indexed by } B \text{ are linearly independent}\}.$$

That vector matroids are matroids follows from Exercise 2.4.1.

Example 2.4.4 Consider the matrix

$$A = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 1 & 0 \end{pmatrix}.$$

with columns indexed by $S = \{1, 2, 3\}$. The vector matroid of A is $(S, \mathcal{I})$ where $\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}\}$.

For vector matroids it is natural to define a maximal independent set to be a basis. We do this for matroids in general:

Definition 2.4.5 A *basis* of matroid $(S, \mathcal{I})$ is a subset $B \subseteq S$ such that $B \in \mathcal{I}$ but no other superset of B is in $\mathcal{I}$.

Lemma 2.4.6 *All bases of a matroid have the same number of elements.*

Proof. Let A and B be two bases of a matroid $M = (S, \mathcal{I})$. Suppose for contradiction that $|B| > |A|$. Then pick a subset $B' \subseteq B$ with $|B'| = |A| + 1$. By property I2, we have $B' \in \mathcal{I}$. By property I3 there exists $b \in B' \setminus A$ such that $A \cup \{b\} \in \mathcal{I}$. This contradicts A being a maximal independent subset of S. □

The size of any basis of a matroid is called the *rank* of the matroid.

2.4.2 The cycle matroid of a graph

We now consider matroids arising from graphs.

Definition 2.4.7 Let $G = (V, E)$ be a graph. Let $S = E$. We identify subsets of S with spanning subgraphs of G. (By letting a subset $A \subseteq S$ correspond to the spanning subgraph with edge set A.) We let $\mathcal{I}$ be the set of subgraphs not containing a cycle. The pair $(S, \mathcal{I})$ is called the *cycle matroid* of G.

In other words the independent sets of the cycle matroid are the (spanning) subforests of G.

Example 2.4.8 The cycle matroid of the graph on the left in Figure 21 has 14 independent sets:

$$\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\},$$

$$\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}\}$$

with 1 denoting the edge not being part of the cycle.

Proposition 2.4.9 *The cycle matroid of a graph is a matroid.*

Proof. The empty set contains no cycle. Hence $\emptyset \in \mathcal{I}$. If A contains no cycle, then a subgraph of B also contains no cycle. This proves I1 and I2.

Suppose I3 did not hold. Let A and B be two subforests with $|B| = |A| + 1$ and such that for every edge b of B, $A \cup \{b\}$ contains a cycle. Then for every edge of B the two ends are connected in A by a walk. Hence the numbers of components satisfy $c(A) \leq c(B)$. Since A and B are forests $c(A) = n - |A| > n - |B| = c(B)$. This is a contradiction. □

In the same way that we defined bases for matroids in general, we will now define cycles for matroids. However, it is common not to call these cycles "cycles" but rather "circuits".

Definition 2.4.10 A minimal dependent set of a matroid $M = (S, \mathcal{I})$ is called a *circuit*. The set of all circuits is denoted by $\mathcal{C}(M)$.

Example 2.4.11 The cycle matroid of Example 2.4.8 has only a single circuit, namely $\{2, 3, 4\}$.

Example 2.4.12 The circuits of Exercise 2.0.1 are

$$\mathcal{C}(M) = \{\{c\}, \{b, d\}, \{a, b, e\}, \{a, d, e\}\}.$$

Exercise 2.4.13 Let G be the graph in Example 2.4.8. Let H be a spanning subgraph of G with isolated vertices removed. Prove that H is cycle if and only if it is a circuit of the cycle matroid of G. Prove the same statement for an arbitrary graph G.

Exercise 2.4.14 Let G be a connected graph. Prove that the spanning trees of G are exactly the bases of the cycle matroid of G.

Exercise 2.4.15 Consider the matrix

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

with columns indexed by $S = \{1, 2, 3, 4, 5, 6, 7\}$. Make a drawing of the vectors. (Draw how the rays they generate intersect the triangle $\text{conv}(e_1, e_2, e_3)$.) Find all independent sets of the the vector matroid of A. Is $\{4, 5, 6\}$ an independent set? Consider the pair $(S, \mathcal{I}')$ where $\mathcal{I}' := \mathcal{I} \setminus \{\{4, 5, 6\}\}$. Prove that $(S, \mathcal{I}')$ is a matroid.

(Hint: Instead of working over $\mathbb{R}$, consider the field $\mathbb{Z}/2\mathbb{Z}$.[16] Over this field A will still define a vector matroid and the argument of Exercise 2.4.1 still holds. Prove that the resulting vector matroid is $(S, \mathcal{I}')$.)

The matroid $(S, \mathcal{I}')$ in Exercise 2.4.15 is called the Fano matroid. If is not realisable as a vector matroid over $\mathbb{R}$, but is over $\mathbb{Z}/2\mathbb{Z}$.

2.5 The transversal matroid of a bipartite graph

So far we have seen how to construct the vector matroid of a vector configuration and the cycle matroid of a graph. In this section we will see a way to construct a matroid from a bipartite graph.

[16] As explained in the Linear Algebra class, concepts like linear independence, linear subspaces, determinants and row reduction work for fields different from the well-known $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$. An example of such different field is $\mathbb{Z}/2\mathbb{Z}$, having just two elements $\{0, 1\}$. The field operations are as usual with the exceptions that $1 + 1 = 0$ and $-1 = 1$.

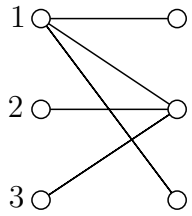

Figure 29: The graph used in Example 2.5.2.

Definition 2.5.1 Let $G[X, Y]$ be a bipartite graph. The transversal matroid of G is the pair $(X, \mathcal{I})$ where

$$\mathcal{I} := \{I \subseteq X : \exists \text{ a matching } M : I \text{ is the set of vertices of } X \text{ covered by } M\}.$$

Example 2.5.2 Consider the graph $G[X, Y]$ in Figure 29. The transversal matroid of G has the independent sets

$$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}\}.$$

Proposition 2.5.3 *The transversal matroid $(X, \mathcal{I})$ of a bipartite graph $G[X, Y]$ is a matroid.*

Proof. We must prove that $\mathcal{I}$ satisfies matroid axioms I1, I2 and I3. Choosing the empty matching $M =$, we see that $\emptyset \in \mathcal{I}$. This proves I1.

Because any subset of a matching in $G[X, Y]$ is also a matching, any subset of an element $I \in \mathcal{I}$ is also a subset in $\mathcal{I}$. This proves I2.

Finally, to prove I3, let $A, B \in \mathcal{I}$ with $|B| = |A| + 1$ and let M_A and M_B be corresponding matchings in G. Then $|M_B| = |M_A| + 1$. Consider the graph G' being G restricted to the vertices $A \cup B$ and Y. The matching M_A is not a maximum matching in G' because M_B has larger cardinality. By Berge' Theorem 1.2.3 there exists an M_A-augmenting path P in G'. Consequently $M_A \triangle P$ is a matching in G' covering exactly A and one vertex b from $B \setminus A$. Hence we have found an element $b \in B \setminus A$ such that $A \cup b \in \mathcal{I}$, as desired. □

2.6 Basis axioms

We observe that if the bases of matroid are known, it is easy to find the independent sets. Namely, a set is independent if it is a subset of a basis. It is desirable to characterise matroids in terms of their bases.

Theorem 2.6.1 *Let $M = (S, \mathcal{I})$ be a matroid. The set $\mathcal{B}$ of bases of M satisfies:*

B1: $\mathcal{B} \neq \emptyset$ and no element from $\mathcal{B}$ is a subset of another element from $\mathcal{B}$.

B2: If $B_1, B_2 \in \mathcal{B}$ and $x \in B_1$ then there exists $y \in B_2$ such that $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{B}$.

Conversely, if a collection of subsets $\mathcal{B}$ satisfies B1 and B2 above, then

$$\mathcal{I} = \{I \subseteq S : \exists B \in \mathcal{B} : I \subseteq B\}$$

is the independent sets of a matroid (with set of bases equal to $\mathcal{B}$).

Proof. Because $\emptyset \in I$, there is a at least one basis of M. That means $\mathcal{B} \neq \emptyset$. Because bases have the same size (Lemma 2.4.6), no two different bases can be contained in each other. This proves B1.

For B2, let B_1 and B_2 be bases and $x \in B_1$. Then $B_1 \in \mathcal{I}$ implies $B_1 \setminus \{x\} \in \mathcal{I}$. By I3 there exists $y \in B_2 \setminus (B_1 \setminus \{x\})$ so that $(B_1 \setminus \{x\}) \cup \{y\} \in \mathcal{I}$. This independent set must be a subset of a basis. But since every basis has the same size, and $(B_1 \setminus \{x\}) \cup \{y\}$ has this size, $(B_1 \setminus \{x\}) \cup \{y\}$ must be a basis.

Suppose now that $\mathcal{B}$ is a collection of subsets of a set S satisfying B1 and B2 and that $\mathcal{I}$ is defined as above. We want to prove that $(S, \mathcal{I})$ is a matroid.

Because $\mathcal{B} \neq \emptyset$, we have $\emptyset \in \mathcal{I}$, proving I1. I2 follows just by looking at the definition of $\mathcal{I}$ — if I satisfies $\exists B \in \mathcal{B} : I \subseteq B$ then so does any subset of I.

We now prove that all elements of $\mathcal{B}$ have the same size. Suppose $B_1, B_2 \in \mathcal{B}$ with $|B_2| > |B_1|$. Then we can repeatedly apply B2 until we reach a B_1' with $B_1' \subseteq B_2$ and $|B_1'| < |B_2|$. This contradicts second part of B1.

To prove I3, let $X, Y \in \mathcal{I}$ with $|Y| = |X| + 1$. We must have $X \subseteq B_1 \in \mathcal{I}$ and $Y \subseteq B_2 \in \mathcal{I}$ for some B_1 and B_2. Suppose the members of sets were as follows:

$$X = \{x_1, \ldots, x_k\}$$

$$B_1 = \{x_1, \ldots, x_k, b_{k+1}, \ldots, b_r\}$$

$$Y = \{y_1, \ldots, y_{k+1}\}$$

$$B_2 = \{y_1, \ldots, y_{k+1}, b'_{k+2}, \ldots, b'_r\}$$

Choose an element $b \in B_1 \setminus X$. Apply B2 to get a $y \in B_2$ such that $(B_1 \setminus \{b\}) \cup \{y\} \in \mathcal{B}$. If $y \in Y$ then $X \cup \{y\} \subseteq (B_1 \setminus \{b\}) \cup \{y\}$ and $X \cup \{y\} \in \mathcal{I}$ as desired. If $y \notin Y$ then keep replacing elements of X by picking another $b \in B_1 \setminus X$ and replace with another found y. Eventually, because there are more b's from B_1 then from B_2, the y must be in Y. When that happens, we conclude that $X \cup \{y\} \in \mathcal{I}$. □

Two proofs of the same matroid statement are often close to identical. For example, the proof above is very similar to the proof of [11, Theorem 10.7]. Indeed the names of the symbols have been taken from there.

2.7 Matroid polytopes

Exercise 2.7.1 Consider the graph G in Figure 30. Let e be the edge 5.

- Find all bases of the cycle matroid of G.
- Find all bases of the cycle matroid of G/e (contraction of e in G).
- Find all bases of the cycle matroid of $G \setminus e$ (deletion of e from G).

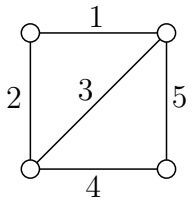

Figure 30: The graph used in Example 2.7.1.

- What is the relation among the number of bases of the matroids above?

Definition 2.7.2 ([2]) Let $M = (S, \mathcal{I})$ be a matroid with set of bases $\mathcal{B}$. Define the *matroid polytope* of M

$$P(M) := \text{conv}(\chi_B) : B \in \mathcal{B}\}$$

where $\chi_B \in \{0,1\}^S$ is the characteristic vector of $B \subseteq S$.

Example 2.7.3 The matroid polytope of the cycle matroid of a 3-cycle is a triangle in $\mathbb{R}^3$.

How would we check if the convex hull of the columns of

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

is a matroid polytope? The theorem below gives an easy characterisation of matroid polytopes. For low dimensions, to see if a polytope is a matroid polytope we simply draw the polytope and look at its edges. The polytope above is a matroid polytope.

Theorem 2.7.4 ([2]) *A non-empty polytope $P \subseteq \mathbb{R}^n$ is a matroid polytope if and only if*

1. *its vertices are in $\{0,1\}^n$ and*
2. *its edges are all in directions $e_i - e_j$.*

Proof. $\Rightarrow$:Suppose P is a matroid polytope. Then all vertices of P are among the set of points we take convex hull of in the definition of the matroid polytope. This proves the first claim. To prove the second claim, let $I, J \in \mathcal{B}$ so that χ_I and χ_J are connected by an edge. Define $m = \frac{1}{2}(\chi_I + \chi_J)$. We give the elements of S names $1, \ldots$. Without loss of generality we can write up

$$\begin{aligned} \chi_I = &\quad (1,\ldots,1, \quad 0,\ldots,0, \quad 1,\ldots,1, \quad 0,\ldots,0) \\ \chi_J = &\quad (0,\ldots,0, \quad 1,\ldots,1, \quad 1,\ldots,1, \quad 0,\ldots,0) \end{aligned}$$

Where we let $p \in \mathbb{N}$ be such that $2p$ is the number of positions at which the two vectors differ. The number p must be an integer because all bases of a matroid have the same size (Lemma 2.4.6).

Our goal is to prove $p = 1$. Suppose for contradiction that $p > 1$. We then use $B2$ to exchange $1 \in I$ with some element of J. Without loss of generality we may assume $K_1 := (I \setminus \{1\}) \cup \{p+1\} \in \mathcal{B}$.

Now exchange $p+1 \in J$ with an element from I to get K_2. Now $K_2 \neq (J \setminus \{p+1\}) \cup \{1\}$ because if it was this vector $\frac{1}{2}(\chi_{K_1} + \chi_{K_2}) = m$ contradicting that m is on an edge. Therefore (without loss of generality) $K_2 = (J \setminus \{p+1\}) \cup \{2\}$.

Now exchange $2 \in I$ with an element from J to get K_3. Now $K_3 \neq (I \setminus \{2\}) \cup \{p+1\}$ because if it was this vector $\frac{1}{2}(\chi_{K_2} + \chi_{K_3}) = m$ contradicting that m is on an edge. Therefore (without loss of generality) $K_3 = (I \setminus \{2\}) \cup \{p+2\}$.

Now exchange $p+2 \in J$ with an element from I to get K_4. Now $K_4 \neq (J \setminus \{p+2\}) \cup \{1\}$ because if it was this vector $\frac{1}{4}(\chi_{K_1} + \chi_{K_2} + \chi_{K_3} + \chi_{K_4}) = m$ contradicting that m is on an edge. Moreover $K_4 \neq (J \setminus \{p+2\}) \cup \{2\}$ because then $\frac{1}{2}(\chi_{K_3} + \chi_{K_4}) = m$ contradicting that m is on an edge. Therefore (without loss of generality) $K_4 = (J \setminus \{p+2\}) \cup \{3\}$.

Continuing in this way, we will run out of options of elements to introduce to the bases. That will be a contradiction. Therefore the assumption that $p > 1$ is wrong, and p must be equal to 1, implying the second statement.

$\Leftarrow$: Conversely, let P be a polytope with the two properties and coordinates indexed by a set S. We want to construct a matroid M so that $P(M) = P$. Define

$$\mathcal{B} = \{I \subseteq S : \chi_I \in P\}$$

It suffices to prove that $\mathcal{B}$ is the set of bases of a matroid. By Theorem 2.6.1 it suffices to prove B1 and B2.

Because $P \neq \emptyset$, we have $\mathcal{B} \neq \emptyset$. Moreover, because any two vertices of a polytope are connected by an edge path, and edges are in directions $e_i - e_j$, all vertices of P have the same coordinate sums. That proves that all elements in $\mathcal{B}$ have the same number of elements. Hence B1 is satisfied.

To prove B2, let $I, J \in \mathcal{B}$ and $x \in I$. If $x \in J$, we can simply choose $y = x$ and satisfy B2. Hence we assume that $x \notin J$. After rearranging coordinates we have

$$\begin{array}{rcccccc} \chi_I = & (1,\ldots,1, & 0,\ldots,0 & 1,\ldots,1, & 0,\ldots,0) \\ \chi_J = & (0,\ldots,0, & 1,\ldots,1, & 1,\ldots,1, & 0,\ldots,0) \\ & \quad A & \quad B & \quad C & \quad D \end{array}$$

with A, B, C, D being a partition of S into four subsets. We have $x \in A$, $I = A \cup C$ and $J = B \cup C$. Let $E_1, \ldots, E_r$ denote a *subset* of the edge directions in P leaving vertex χ_I allowing the expression $\chi_J - \chi_I = \sum_i a_i E_i$ with all $a_i > 0$. Having $\chi_I \in P \subseteq [0,1]^S$ implies for $j = 1, \ldots, r$

- $(E_j)_i \geq 0$ for $i \in B \cup D$ and
- $(E_j)_i \leq 0$ for $i \in A \cup C$.

Consequently, because $\chi_J - \chi_I$ is zero at coordinates indexed by D, we conclude that all E_j have D-coordinates 0.

Because $(\chi_J - \chi_I)_x = -1$, some E_j must equal $e_y - e_x$ for some y. By the second inequality above, $y \in B \cup D$. But also $y \notin D$ because all D-coordinates are zero. Hence we have found $y \in B = J \setminus I$ such that $(I \setminus \{x\}) \cup \{y\} \in \mathcal{B}$. □

Exercise 2.7.5 The *diameter* of a graph G is the largest distance between two vertices of the graph. That is, it is the largest entry of the distance matrix of the matrix of G, assuming that all edges are assigned length 1. The *edge graph* of a polytope P is the graph with V being the vertices of P and two edge being connected by an edge if their convex hull is an edge in P. Prove that the diameter of the edge graph of a matroid polytope is at most the rank of the matroid. (Hint: Let χ_M and $\chi_{M'}$ be two vertices and use Theorem 2.6.1 to construct a sequence of vertices between them such that the convex hull of two consecutive vertices is an edge of P.)[17]

2.7.1 Duals, deletion and contraction

An immediate consequence of Theorem 2.7.4 is that if P is a matroid polytope, then so is $(1, \ldots, 1)^t - P = \{(1, \ldots, 1)^t - v : v \in P\}$. This allows us to define the dual matroid.

Definition 2.7.6 Let M be a matroid on S with matroid polytope P then the *dual* matroid M^* is the matroid on S with matroid polytope $\{(1, \ldots, 1)^t - v : v \in P\}$.

We note that B is a basis of $M = (S, \mathcal{I})$ if and only if $S \setminus B$ is a basis of M^*.

Example 2.7.7 The dual matroid of the cycle matroid of the three cycle C_3 has independent sets $\{\emptyset, \{1\}, \{2\}, \{3\}\}$.

Exercise 2.7.8 Do the dual graphs of the two plane graphs in Figure 27 have equal cycle matroids? (Hint: How do the cycles of the dual graphs look? Can we say which edges are involved in a cycle without refering to the embedding of the graph, but only to the combinatorics?)

Recall that a supporting hyperplane $H \subseteq \mathbb{R}^n$ for a polytope $P \subseteq \mathbb{R}^n$ is a hyperplane touching P in a non-empty set an having the rest of P contained on one side of H. The non-empty intersection $F = P \cap H$ is called a *face* of P.

It follows from convex geometry that if P is a zero-one polytope (every vertex of P has vertices in $\{0, 1\}^n$) then so is any face F of P. Moreover, each edge of F is also an edge of P. Consequently, each face F of a matroid polytope P is a matroid polytope.

Definition 2.7.9 Let $M = (S, \mathcal{I})$ be a matroid.

- An element $e \in S$ is called a *loop* if e is not contained in any basis of M.
- An element $e \in S$ is called a *coloop* if e is contained in every basis of M.

[17] In general the Hirsch conjecture says that the diameter of a D-dimensional polytope described by m inequalities is $\leq m - D$. This conjecture was disproved by Santos in 2010.

Notice that if a graph G contains a loop e at a vertex v, then $\{e\}$ is a dependent set in the cycle matroid of G and therefore e cannot be in any basis. Consequently, e is a loop in the cycle matroid of G.

Definition 2.7.10 Let $M = (S, \mathcal{I})$ be a matroid with matroid polytope P and a $i \in S$. Let $\pi : \mathbb{R}^S \to \mathbb{R}^{S \setminus \{i\}}$ be the coordinate projection leaving out the ith entry. Then for $i \in S$ not a coloop we define

- the deletion matroid $M \setminus i$ to be the matroid on $S \setminus \{i\}$ with polytope $\pi(P \cap \{x \in \mathbb{R}^S : x_i = 0\})$.

For $j \in S$ not a loop we define

- the contraction matroid M/i to be the matroid on $S \setminus \{i\}$ with polytope $\pi(P \cap \{x \in \mathbb{R}^S : x_i = 1\})$.

Because the faces $P \cap \{x \in \mathbb{R}^S : x_i = 0\}$ and $P \cap \{x \in \mathbb{R}^S : x_i = 1\}$ both are matroid polytopes (and because these polytopes are "flat"), their projections are also matroid polytopes. Consequently, the deletion and contraction matroids are well-defined.

For cycle matroids of graphs, the operations of deletion and contraction correspond to deletion and contraction of edges in the graphs. This is illustrated by Exercise 2.7.1. The cycle matroid G/e is the contraction matroid of the cycle matroid of G, while the cycle matroid $G \setminus e$ is the deletion matroid of the cycle matroid of G.

Exercise 2.7.11 How are the rank of a matroid M and the rank of M^* related?

Exercise 2.7.12 How are the ranks of matroids M, $M \setminus i$ and M/i related?

Exercise 2.7.13 If we for any graph G let $M(G)$ denote the cycle matroid of G, is it then true for any plane graph G and $e \in E(G)$ (not a loop or coloop) that

- $M(G \setminus e) = M(G) \setminus e$?
- $M(G/e) = M(G)/e$?
- $M(G)^* = M(G^*)$?

2.8 Greedy algorithms for independence systems

Given a connected graph G with a weight function $w : E \to \mathbb{R}$ we are interested in finding a spanning tree T of G with maximal weight $w(T)$. This can be accomplished with a *greedy algorithm*.

Algorithm 2.8.1 (Kruskal's Algorithm)
Input: *A connected graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$.*
Output: *A spanning tree T of G with maximal w weight.*

- *Let $T := \emptyset$.*

- *While T is not a spanning tree*
  - *Find $e \in E \setminus E(T)$ such that $T \cup \{e\}$ does not contain a cycle and $w(e)$ is maximal.*
  - $T := T \cup \{e\}$.
- *Return T*

The above algorithm should be well-known from your Mathematical Programming class. Typically the algorithm is formulated so that it finds a *min-weight spanning tree*, but that is not an essential difference.

With our knowledge of matroid polytopes we now observe that we have bijections between the three sets:

- the spanning trees of G
- the bases of the cycle matroid of G
- the vertices of the matroid polytope of the cycle matroid of G

In fact, we could also have found a max-weight spanning tree of G by maximising w over the matroid polytope of G, using for example the simplex method. However, just as for the matching polytope (Theorem 1.7.6), we might not have a short inequality description of the polytope, so that will not be our main point.

Rather, we observe that the max-weight spanning tree problem reduces to the following problem:

- Find an max w-weight independent set of a given matroid.

Namely, if w is positive, a solution to the above problem will be basis. Moreover, we can by adding a constant to all coordinates of w assume that the w is positive without changing which spanning trees have maximal weight.

The algorithm for solving the matroid problem is a straight forward translation of Kruskal's Algorithm into matroid language.

Algorithm 2.8.2 (The greedy algorithm for matroids)
Input: *A matroid M with ground set E and a function $w : E \to \mathbb{R}$.*
Output: *An independent set B of M with maximal w-weight.*

- *Let $I := \emptyset$.*
- *While* ($\exists e \in E \setminus I : I \cup \{e\} \in \mathcal{I}$ and $w(e) > 0$)
  - *Find $e \in E \setminus I$ such that $I \cup \{e\}$ is independent and $w(e)$ is maximal among such choices.*
  - $I := I \cup \{e\}$.
- *Return I*

For the algorithm to run, we do not actually need to know the set $\mathcal{I}$ of independent sets of M. Rather, it suffices to have an algorithm which will check whether a given set I is independent or not in M. In particular we are then able to check the condition in the while loop.

Proof. The algorithm terminates because $|I|$ increases in each iteration. We now prove correctness. For this we may assume that $w \in \mathbb{R}^E_{>0}$, since all e with $w(e) \leq 0$ are ignored by the algorithm. So indeed, the algorithm just operates with the (repeated) deletion matroid of M where all negative weight elements of the ground set E have been deleted.

The following statement holds after each iteration of the algorithm:

- There exists a basis $B \in \mathcal{I}$ with $I \subseteq B$ and $w(B)$ maximal among all bases of M.

This is true at the beginning of the algorithm because $I = \emptyset$ and we can take B to be any basis of M. For the induction step, let e be the element which is being added to I. If $e \in B$, then the same choice of B works for the next iteration (because $I \cup \{e\} \subseteq B$ and B has maximal w-weight).

If $e \notin B$ we need to construct a new basis B' containing $I \cup \{e\}$. If $B' := I \cup e$ is not already a basis then $|B'| < |B|$ and we can pick a $C \subseteq B$ with $|C| = |B'| + 1$. By I2, C is independent. Applying I3 to B' and C we get an element that we may add to B' keeping it independent. If the new B' is still not a basis, we have $|B'| < |B|$, can pick a new C and repeat the argument. Eventually $|B'| = |B|$ with $I \cup e \subseteq B'$ and B' independent. Hence $B' \neq B$ is a basis and we must show that it has maximal weight. Because both B and B' are bases they have the same size and by construction only $|B| + 1$ elements are involved in total. Therefore $B' \setminus B = \{e\}$. By counting cardinalities, $B \setminus B' = \{j\}$ for some j. Hence $B' = B \setminus \{j\} \cup \{e\}$.

Notice that $w(j) \leq w(e)$. (If we had $w(j) > w(e)$ we would have included j in I earlier because $j \in B \supseteq I$ and $j \notin B' \supseteq I$, making $I \cup \{j\}$ independent. This contradicts $j \notin I$.)

We conclude:

$$w(B') = w(B) + w(e) - w(j) \geq w(B).$$

Hence B' works as a choice of independent set containing $I \cup \{e\}$. (Hint: transversal matroids.)

When the algorithm terminates, it is because I is a basis. But then $I \subseteq B$ with the weight of B being maximal. Because we have no strict inclusion among bases, $I = B$. This is an independent set of maximal weight. □

Exercise 2.8.3 In what sense is Algorithm 1.3.9 a greedy matroid algorithm? (Hint: transversal matroids.)

By an *independence system* with ground set S we mean a set $\mathcal{I}$ of subsets of S, such that matroid axioms I1 and I2 are satisfied. An example is the set of matchings in a graph.

We could try to apply the greedy algorithm to any independence system. However, that will not always work. For example the set of matchings of the graph to the left in Figure 11 is an independence system. Applying the greedy algorithm would wrongly produce a matching with weight 3. We conclude that the independence system is not a matroid.

More surprising is the following theorem.

Theorem 2.8.4 *[11, Theorem 10.34] Let $M = (S, \mathcal{I})$ be an independence system. If for every function $w : S \to \mathbb{R}_{\geq 0}$ the greedy Algorithm 2.8.2 produces an independent set with maximal w-weight, then M is a matroid.*

Proof. It suffices to prove I3. Let A, B with $|B| = |A| + 1$. Choose x so that $0 \leq \frac{|A|-|A\cap B|}{(|A|-|A\cap B|)+1} < x < 1$ and define $w \in \mathbb{R}^S$ as follows:

$$w_i := \begin{cases} 1 & \text{for } i \in A \\ x & \text{for } i \in B \setminus A \\ 0 & \text{otherwise} \end{cases}$$

Running Algorithm 2.8.2 with weights w we will first pick the elements in A. Suppose now for contradiction that there is no $b \in B \setminus A$ such that $A \cup \{b\} \in \mathcal{I}$. Then the algorithm will return A. We have $w(A) = |A|$ and $w(B) = |A \cap B| + (|A| + 1 - |A \cap B|)x$. Consequently,

$$w(A)-w(B) = |A|-|A\cap B|-(|A|+1-|A\cap B|)x < |A|-|A\cap B|-(|A|-|A\cap B|) = 0,$$

where the inequality follows from $|A| - |A \cap B| < x((|A| - |A \cap B|) + 1)$. This contradicts the algorithm always picking a max-weight independent set. □

2.9 Rado's generalisation of Hall's Theorem

For a matroid $(S, \mathcal{I})$ of a matrix, there is a natural notion of rank of a subset $I \subseteq S$, namely the rank the submatrix with columns indexed by I. We generalise this notion to any matroid.

Definition 2.9.1 Let $(S, \mathcal{I})$ be a matroid. We define the rank $\rho(I)$ of a subset $I \subseteq S$ as

$$\rho(I) = \max_{A \subseteq I : A \in \mathcal{I}} |A|.$$

We wish to make a generalisation of Hall's theorem for matroids.

Definition 2.9.2 Let $G[X, Y]$ be a bipartite graph. The *deficiency* is defined as

$$\sigma(G) = \max_{A \subseteq X} (|A| - |N(A)|).$$

By taking $A = \emptyset$ we see that the deficiency is always non-negative.

Remark 2.9.3 Notice that if we add a vertex to Y and introduce edges from it to all vertices of X, then the deficiency drops by 1.

We will prove that the deficiency is what we lack from having all vertices in X matched. The following is a strong version of Theorem 1.2.20.

Theorem 2.9.4 *The matching number of a bipartite graph satisfies*

$$\alpha'(G) = |X| - \sigma(G)$$

Proof. For every set A we have $\alpha'(G) \leq |X| - (|A| - |N(A)|)$. This proves

$$\alpha'(G) \leq |X| - \sigma(G).$$

For the other inequality we must prove that G has a matching of size $|X| - \sigma(G)$. Let G' be $G[X, Y]$ but where we have repeatedly $\sigma(G)$ times added a new vertex to Y as in Remark 2.9.3. The deficiency of G' is zero by the remark, implying $|A| - |N(A)| \leq 0$ for every $A \subseteq X$. By Hall's Theorem 1.2.20, there exists a matching M in G' covering all vertices of X. Restricted to G, this matching will have size at least $|X| - \sigma(G)$. This proves $\alpha'(G) \geq |X| - \sigma(G)$. □

Exercise 2.9.5 How would you deduce Hall's Theorem 1.2.20 from Theorem 2.9.4?

We may rephrase the statement of Theorem 2.9.4 as

$$\alpha'(G) = \min_{A \subseteq X}(|X| + |N(A)| - |A|)$$

which we will now generalise to matroids.

Let $S_1, \ldots, S_m$ be a partition of S. A partial system of representatives (p.s.r.) is a subset of S such that it contains at most one element from each S_i. The set of partial representatives is a matroid.

Theorem 2.9.6 (Rado) *Let $M = (S, \mathcal{I})$ be a matroid. The maximal size of an M-independent p.s.r. is*

$$\min_{A \subseteq \{S_1, \ldots, S_m\}}(|S| + \rho(\bigcup A) - |A|).$$

where $\bigcup A$ is the union of sets in A.

We will not prove Rado's Theorem. Rather, we observe that Hall's theorem is a special case of Rado's. Given a graph $G[X, Y]$ with edge set E, take $S = E$ and partition S into $|X|$ groups according to which vertex of X the edges are incident to. Take $M = (S, \mathcal{I})$ to be the matroid where a subset of edges is independent if each vertex of Y is covered at most once by the set. Then $\rho(\bigcup A) = |N(A)|$. Finally, a partial system of representatives is independent in M if and only if it is a matching in $G[X, Y]$. This explains how Hall's Theorem 2.9.4 follows from Rado's Theorem.

2.10 Matroid intersection

Exercise 2.10.1 Let S be the disjoint union of $S_1, \ldots, S_k$ and let $d_1, \ldots, d_k \in \mathbb{N}$. Let $I \subseteq S$ be independent if for all i, $|I \cap S_i| \leq d_i$. Prove that this defines a matroid. This matroid is called a *partition matroid*.

Exercise 2.10.2 Let $G[X, Y]$ be bipartite graph with edges E. We say that $I \subseteq E$ is independent if each $x \in X$ is covered by at most one edge from I. Prove that this defines a partition matroid.

Exercise 2.10.3 Given a bipartite graph $G[X, Y]$ with edges E. Is the set of matchings in G a matroid on the ground set E?

Let $G[X, Y]$ be a bipartite graph with edges E. We observe that $M \subseteq E$ is a matching if and only if it is independent in the matroid of Exercise 2.10.2 and in the matroid of Exercise 2.10.2 where Y is considered instead of X. However, these matchings do not form the independent sets of a matroid by Exercise 2.10.3. *We conclude that the intersection of two matroids is not a matroid.*

What is meant by matroid intersection of two matroids $(S, \mathcal{I}_1)$ and $(S, \mathcal{I}_2)$ is finding a set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ with largest cardinality $|I|$ (or if a weight function w is given, one with largest weight $w(I)$). By the observations in the paragraph above, if we are able to find such largest sets in the intersection of two partition matroids, we can easily solve the maximum bipartite matching problem. Other applications of matroid intersection are given in the following subsections.

2.10.1 Matroid intersection is difficult (NP-hard) for three matroids

A Hamiltonian path in a G graph is subgraph of G being a path involving all vertices of G. Similarly a Hamiltonian path in a directed graph G is a directed path in G involving all vertices. Deciding if a graph (directed or undirected) has a Hamiltonian path is difficult. (It is NP-hard).

If we were able to do matroid intersection of three matroids quickly, then the following algorithm could be used to check for directed paths.

Algorithm 2.10.4
Input: *A directed graph $G = (V, E)$ and vertices $s \neq t$*
Output: *"yes" if there exists a directed path from s to t of length $|V| - 1$ and "no" otherwise*

- *Construct a matroid $M_1 = (E, \mathcal{I}_1)$ being the partition matroid with independent sets being subsets of E with at most zero outgoing edges from t and at most 1 outgoing edge from each of the vertices in $V \setminus \{t\}$.*
- *Construct a matroid $M_2 = (E, \mathcal{I}_2)$ on E being the partition matroid with independent sets being subsets of E with at most zero in-going edges to s and at most 1 in-going edge to each of the vertices in $V \setminus \{s\}$.*
- *Let $M_3 = (E, \mathcal{I}_3)$ be the cycle matroid of the underlying undirected graph of G.*

- *Let I be a set of largest size in $\mathcal{I}_1 \cap \mathcal{I}_2 \cap \mathcal{I}_3$.*
- *If $|I| = |V| - 1$ then return "yes", otherwise return "no".*

In the algorithm we would not actually store $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$, but rather present them by oracles.

Exercise 2.10.5 Prove that the algorithm is correct.

Exercise 2.10.6 How do we decide if a set is independent in each of the three matroids above?

Exercise 2.10.7 How would you use Algorithm 2.10.4 to decide if a (undirected) graph has a Hamiltonian path?

Exercise 2.10.8 Can we make a similar algorithm checking if a graph has a Hamiltonian cycle?

We conclude that matroid intersection for three matroids is difficult. (It is NP-hard.)

2.10.2 Matroid intersection for two matroids

Edmonds proved that matroid intersection of two matroids can be done in polynomial time. We will not present an algorithm here but rather refer to [10], where the matroid intersection of two matroids is reduced to finding a partial system of representatives (in the sense of Rado's Theorem 2.9.6) for a partition and a matroid. That problem is then shown to have a polynomial time algorithm.

Another reason for not presenting a matroid intersection algorithm here is that that is done in the *mixed integer programming* class.

Remark 2.10.9 A final comment on matroid intersection is that the problem is closely related to the famous P≠NP problem. If there is a polynomial time algorithm for intersecting three matroid, then P=NP. We have not defined what P and NP are. Neither were these classes defined when Edmonds first asked whether the travelling salesman problem could be solved in polynomial time. Cook stated the precise P≠NP conjecture in 1971.

Exercise 2.10.10 What is the travelling salesman problem? What kind of (weighted) matroid intersection algorithm would be needed to solve it?

Exercise 2.10.11 In a graph G where the edges have colours, how would you decide, using matroid intersection of two matroids, whether there is a spanning tree of G with exactly 5 yellow, 4 red and 3 blue edges (and possibly edges of other colours)? (Hint: use a cycle matroid and a partition matroid with ground set $E(G)$).

Exercise 2.10.12 In Exercise 1.1.3 we observed that it is not obvious if class scheduling with more than one class can be solved by solving matching problems. Is it possible to solve the multi-class scheduling problem using (weighted) matroid intersection problems of three partition matroids, if the students for example should attend six activities during the week?

2.10.3 Completing this section....

If we have more time, our section on matroid intersection can be completed in one of two ways:

Matroid partitioning Edmonds' matroid intersection cardinality algorithm can be realised by solving the matroid partitioning problem.

Reduction of Hamiltonian path A result by Fukuda, Liebling and Margot (1997) shows that it is possible to solve the Hamiltonian path problem by solving an "Optimal vertex in a polyhedron"-problem, which strangely enough looks a lot like a linear programming problem. How can that be?

2.11 A max cardinality matroid intersection algorithm

Recall that the intersection of two matroids might not be a matroid.

Example 2.11.1 If we have two matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$ given by independence oracles we are interested in finding $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|I|$ largest. Our approach will be as in Egerváry's Algorithm 1.3.9. That means that we start with a set I with $|I| = 0$ and try to make it one larger at a time by making a symmetric difference with some set.

Suppose $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ and we have some $a \in S \setminus I$ such that $I \cup \{a\} \in \mathcal{I}_1$. If $I \cup \{a\}$ is also independent in M_2, then we have found an common independent set with $|I| + 1$ elements as desired. If $I \cup \{a\} \notin \mathcal{I}_2$, then we may look for a sequence of pairs $(a_1, b_1), \ldots, (a_p, b_p)$ where $a_i \notin I \ni b_i$ for all i such that $I \cup \{a\} \setminus \{b_1\} \cup \{a_1\}, \ldots, \setminus \{b_p\} \cup \{a_p\} \in \mathcal{I}_2$. At the same time we wish to ensure that the set is also in $\mathcal{I}_1$.

It turns out that this problem can be solved by finding a directed path in a certain graph. The key step is to observe that we from the information like:

$$I \cup \{a\} \in \mathcal{I}$$

$$I \setminus \{b_1\} \cup \{a_1\} \in \mathcal{I} \text{ and } I \cup \{a_1\} \notin \mathcal{I}$$

can conclude Check this!!!

$$I \cup \{a\} \setminus \{b_1\} \cup \{a_1\} \in \mathcal{I}.$$

From Nemhauser and Wolsey we get the following proposition and provide a new proof:

Proposition 2.11.2 *Let $M = (S, \mathcal{I})$ be a matroid, $I \in \mathcal{I}$ and consider two lists of distinct elements $a_1, \ldots, a_p \notin I$ and $b_1, \ldots, b_p \in I$. Suppose that*

$$\forall i : I \cup \{a_i\} \setminus \{b_i\} \in \mathcal{I}$$

and

$$\forall i, j \text{ with } i < j : I \cup \{a_i\} \setminus \{b_j\} \notin \mathcal{I}$$

then

$$\forall i, j : I \cup \{a_i, \ldots, a_j\} \setminus \{b_i, \ldots, b_j\} \in \mathcal{I}.$$

Proof. The proof is by induction in $j - i$, where we simultaneously with proving our result also prove for valid choices of offsets $0 < d_1 \leq d_2 \leq \cdots \leq d_{i-j+1}$ that

$$I \cup \{a_i, \ldots, a_j\} \setminus \{b_{i+d_1}, \ldots, b_{j+d_{j-i+1}}\} \notin \mathcal{I}. \tag{6}$$

For the induction start we have $i = j$ and the two claims (the statement of the proposition and the claim above) are just restatements of our assumptions.

For the induction step we have $j > i$ and first prove (6) by supposing that

$$J := I \cup \{a_i, \ldots, a_j\} \setminus \{b_{i+d_1}, \ldots, b_{j+d_{j-i+1}}\} \in \mathcal{I}.$$

Then $J \setminus \{a_j\}$ is independent and combined with the assumed $I \cup \{a_{j-1}\} \setminus \{b_{j-1}\} \in \mathcal{I}$ matroid axiom I3 gives

$$K := I \cup \{a_i, \ldots, a_{j-1}\} \setminus \{b_{i+d_1}, \ldots, b_{j+d_{j-i}}\} \cup \{b\} \in \mathcal{I}$$

where b cannot be a_{j-1} and must be one of the removed b's. This, however, contradicts the induction hypothesis (6) (since $b_i \in K$).

For the claim of the theorem, observe $I \cup \{a_i, \ldots, a_{j-1}\} \setminus \{b_i, \ldots, b_{j-1}\} \in \mathcal{I}$ and $I \cup \{a_{i+1}, \ldots, a_j\} \setminus \{b_{i+1}, \ldots, b_j\} \in \mathcal{I}$. Therefore $I \cup \{a_i, \ldots, a_{j-1}\} \setminus \{b_i, \ldots, b_j\} \in \mathcal{I}$ and applying axiom I3 to this and the second set, we get that

$$I \cup \{a_i, \ldots, a_j\} \setminus \{b_i, \ldots, b_j\} \in \mathcal{I} \text{ or } I \cup \{a_i, \ldots, a_{j-1}\} \setminus \{b_{i+1}, \ldots, b_j\} \in \mathcal{I}.$$

The second possibility is impossible by the induction hypothesis. □

In its immediate form the proposition is not applicable to our problem (because it speaks about independent sets of equal number of elements), so we convince ourselves about the following variant:

Corollary 2.11.3 *Let $M = (S, \mathcal{I})$ be a matroid, $I \in \mathcal{I}$ and consider two lists of distinct elements $a_1, \ldots, a_p, a_{p+1} \notin I$ and $b_1, \ldots, b_p \in I$. Suppose that*

$$I \cup \{a_{p+1}\} \in \mathcal{I} \text{ and } \forall i \leq p : I \cup \{a_i\} \setminus \{b_i\} \in \mathcal{I} \text{ and } I \cup \{a_i\} \notin \mathcal{I} \tag{7}$$

and

$$\forall i, j \text{ with } i < j : I \cup \{a_i\} \setminus \{b_j\} \notin \mathcal{I} \tag{8}$$

then

$$I \cup \{a_1, \ldots, a_{p+1}\} \setminus \{b_1, \ldots, b_p\} \in \mathcal{I}.$$

Proof. We apply the proposition to a matroid M' with ground set $S \cup \{b_{p+1}\}$ where b_{p+1} is a new element and a set I is independent in M' if and only if $I \setminus \{b_{p+1}\}$ is independent in M. The sequences $a_1, \ldots, a_{p+1}$ and $b_1, \ldots, b_{p+1}$ now satisfy the proposition. Therefore $I \cup \{a_1, \ldots, a_{p+1}\} \setminus \{b_1, \ldots, b_{p+1}\}$ is independent in M'. Consequently $I \cup \{a_1, \ldots, a_{p+1}\} \setminus \{b_1, \ldots, b_p\}$ is independent in M.□

Exercise 2.11.4 Prove that the matroid M' of the proof is indeed a matroid.

We now associate a graph to the problem of extending $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ to an independent set in both matroids.

We define a directed bipartite graph $G[I \cup \{\beta, \beta'\}, I^c]$. For $\mathcal{I}_1$ we add arcs:

- $(b_i, a_i) \in I \times I^c$ whenever $I \cup \{a_i\} \setminus \{b_i\} \in \mathcal{I}_1$
- $(\beta, a_i) \in \{\beta\} \times I^c$ whenever $I \cup \{a_i\} \in \mathcal{I}_1$

For $\mathcal{I}_2$ we add arcs:

- $(a_i, b_i) \in I^c \times I$ whenever $I \cup \{a_i\} \setminus \{b_i\} \in \mathcal{I}_2$
- $(a_i, \beta') \in I^c \times \{\beta'\}$ whenever $I \cup \{a_i\} \in \mathcal{I}_2$

Example 2.11.5

The key observation now is that if $P = (\beta, a_{p+1}, b_p, \ldots, a_1, \beta')$ is a shortest directed (β, β')-path then the follow holds:

- The conditions of the corollary is satisfied. This is immediate for (7), while (8) follows from P being shortest. Hence $I \triangle \{a_1, b_1, \ldots, a_p, b_p, a_{p+1}\} \in \mathcal{I}_1$.
- Reversing arrows in G would be the same as exchanging $\mathcal{I}_1$ and $\mathcal{I}_2$ in the construction of G. Therefore also $I \triangle \{a_1, b_1, \ldots, a_p, b_p, a_{p+1}\} \in \mathcal{I}_2$.

We conclude that if there is a directed (β, β') path in G then $\mathcal{I}_1 \cap \mathcal{I}_2$ has an element af size $|I + 1|$.

Our discussion leads to the following algorithm:

Algorithm 2.11.6 (Max Cardinality Matroid Intersection)
Input: *A set S and independence oracles for two matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$.*
Output: *A set $I \in \mathcal{I}_1 \cap \mathcal{I}_2$ with $|I|$ maximal.*

- *Let $I := \emptyset$*
- *Repeat the following*
  - *Construct the graph $G[I \cup \{\beta, \beta'\}, I^c]$.*
  - *Find a shortest directed (β, β')-path in G — if none exists terminate the algorithm with output I.*
  - *Let $I := I \triangle (V(P) \setminus \{\beta, \beta'\})$.*

Remark 2.11.7 We have not yet argued for the correctness of the algorithm. Namely, we need a proof that when no (β, β')-path is found, then it is because it is not possible to improve $|I|$. We will not present such proof in these notes.

Remark 2.11.8 We can bound the number of oracle calls needed to build up the graphs by a constant times $|S|^3$. Similarly we could bound the time needed to find the shortest paths.

Remark 2.11.9 There also exists algorithms for the weighted matroid intersection problem.

3 Optimum branching

3.1 Optimum branching

Definition 3.1.1 Let $G = (V, E)$ be a directed graph with a weight function $w : E \to \mathbb{R}$. A subset $B \subseteq E$ is a *branching* if

- B contains no cycle
- each vertex has at most one in-going arc from B.

Our goal is to compute a branching with maximal weight. We call such a branching an *optimum branching*.

Example 3.1.2 Consider the graph in Figure 31. An optimum branching is also shown. It has weight 33. Notice that this is not a max-weight spanning tree in the underlying undirected graph.

The algorithm we present was first discovered by Yoeng-jin Chu and Tseng-hong Liu in 1965 and then independently by Edmonds in 1967. It is often referred to as Edmonds' Algorithm. The presentation given here is based on Karp [5] and partly [11].

Before we can present the algorithm and its proof, we need some definitions and lemmas.

Definition 3.1.3 In the setting above. An arc $e \in E$ is called *critical* if

- $w(e) > 0$ and
- for every other arc e' with the same head as e we have $w(e) \geq w(e')$.

We notice that if e is not a critical arc of G, then it is still possible that we need it for an optimum branching. (Take a directed three-cycle with weight 2 for each arc and additionally an arc of weight 1 going from a vertex outside the cycle to the cycle.)

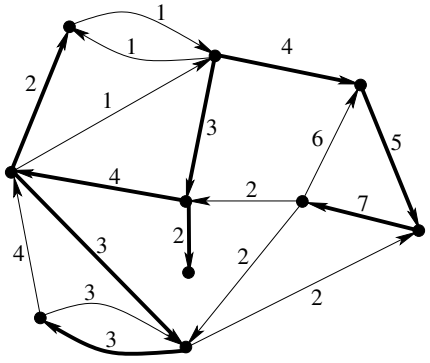

Figure 31: A directed graph with weights. An optimum branching for the graph is indicated with bold edges. See Example 3.1.2.

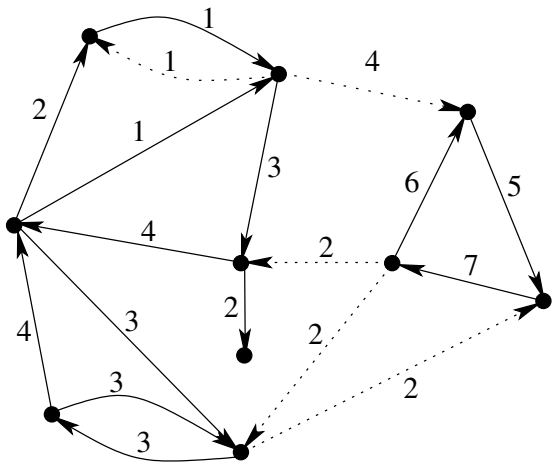

Figure 32: The graph of Example 3.1.4. The non-critical arcs have been removed.

Example 3.1.4 In the example from before. The critical edges are shown in Figure 32.

Notice that while the optimum matching in this case is a tree, it need not be a tree in general. In that case the optimum branching will be a forest.

Definition 3.1.5 A subgraph $H \subseteq E$ is called *critical* if

- each arc of H is critical, and
- no two arcs in H have the same head.

We are in particular interested in *maximal* critical subgraphs.[18]

Example 3.1.6 By leaving out three of the critical edges in our running example, we get a maximal critical subgraph. See Figure 33.

Lemma 3.1.7 *If a maximal critical subgraph H has no cycles, then it is an optimum branching.*

Proof. The subgraph H contains no cycles by assumption and each vertex has in-degree at most 1 because H is critical. We conclude that H is a branching.

We now prove that H is optimum by observing that H chooses a largest weight in-going arc at each vertex. We conclude that H has largest possible weight among all branchings.□

Example 3.1.8 Our maximal critical subgraph contains two directed cycles. Therefore the lemma cannot be applied. Observe that they have no edges in common.

[18]In fact so interested that some people define critical graphs to always be maximal.

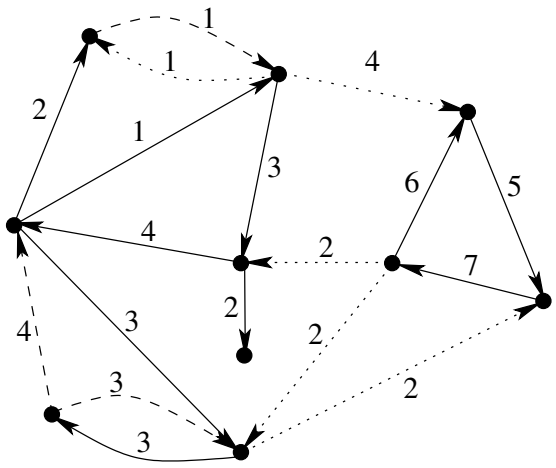

Figure 33: We have removed three critical edges and obtained a maximal critical subgraph.

Lemma 3.1.9 *Let $H \subseteq E$ be a critical subgraph with C_1 and C_2 different subgraph being cycles. Then C_1 and C_2 share no vertices.*

Proof. Suppose C_1 and C_2 shared a vertex v. Then the vertex v_1 right before v in C_1 must be the same as the vertex v_2 right before v in C_2 because only one arc from H has head v. We now continue to $v_1 = v_2$ with the same argument. We conclude, since cycles are finite, that $C_1 = C_2$. □

It is not difficult to prove, that each component of G contains at most one cycle.

Now fix a maximal critical subgraph H of G. Let $C_1, \ldots, C_k$ be the cycles. The following theorem allows us to restrict the search for an optimum branching.

Theorem 3.1.10 *Let H be a maximal critical subgraph. There exists an optimum branching B such that for every C_i we have $|E(C_i) \setminus E(B)| = 1$.*

To not interrupt the presentation of the algorithm we postpone the proof until later.

We now wish to transform G into a graph G' by identifying all vertices in C_1 and replace them by a single vertex u_i. Similarly for $C_2, \ldots, C_k$. We remove loops when doing this identification.

We assign new weight w to the edges of G'.

- For $e \in E(G) \setminus E(C_i)$ if the head of e is in C_i for some i, we let

  $$w'(e) = w(e) - w(\tilde{e}) + w(e_i^0)$$

  where $\tilde{e}$ is the unique edge in C_i with the same head as e and e_i^0 is an edge in C_i with smallest weight.

- For all other e we let $w'(e) = w(e)$.

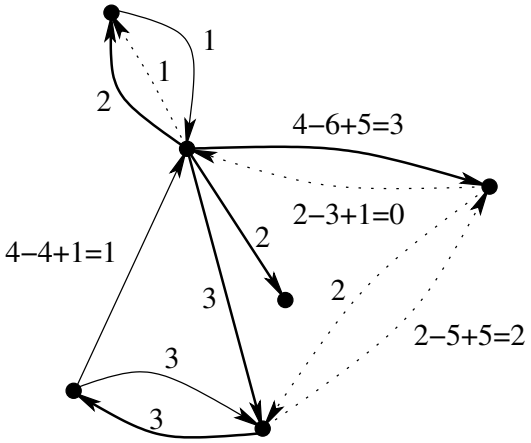

Figure 34: We have identified vertices in the two cycles and updated the weights to obtain G' and w'.

Example 3.1.11 In our running examples we identify the vertices in each of the two cycles and obtain the graph in Figure 34. That some of the edges are dotted should be ignored for now.

Theorem 3.1.12 *Let B be an optimum branching for G with weights w and B' an optimum branching for G' with weights w' as defined above. Then*

$$w(B) - w'(B') = \sum_{i=1}^{k} w(C_i) - \sum_{i=1}^{k} w(e_i^0).$$

Moreover, an optimum branching for G can be obtained from B'.

We postpone the proof until later, and rather illustrate the theorem on our example.

Example 3.1.13 After staring at the graph of Figure 33 we observe that it has an optimum branching B' of w' weight 13. It is indicated with bold edges. The formula of the theorem now holds because

$$33 - 13 = (8 + 18) - (1 + 5).$$

We get a branching in the original graph by for each cycle, extending B' with all but one of the edges in C_1 and all but one of the edges in C_2. We do this in such a way that for the first cycle C_1 with no in-going edges in B', we add all but the smallest weight edge of C_1. For cycle C_2 we add the edges which do not have head equal to the vertex of C_2 already being a the head of an arc in B'.

This gives rise to the algorithm

Algorithm 3.1.14 (Optimum branching)
Input: *A graph G with a weight function $w \to \mathbb{R}$.*
Output: *An optimum branching B in G.*

- *Find the subgraph K of critical arcs.*
- *Let H be a maximal critical subgraph of K obtained by dropping an arc whenever two arcs have the same head.*
- *If H contains no cycles*
  - *then return $B := H$.*

  else
  - *Find the cycles $C_1, \ldots, C_n$ in H.*
  - *Produce the graph G' and w' as defined above (u_i's are new vertices).*
  - *Recursively compute an optimum branching B' on the smaller graph G' with weights w'.*
  - *Construct B by taking all edges in B'. Besides these we take for each C_i all arcs except one:*
    - *If no arc of B' ends at u_i leave out a lowest w-weight arc from C_i.*
    - *else let $(a, b) \in E(G)$ denote the arc inducing the arc ending at u_i. Leave out the arc of C_i with head b.*
  - *Return B.*

The correctness of the algorithm follows from Theorem 3.1.12.

Exercise 3.1.15 Complete the computation of the optimum branching of Example 3.1.2 by running the algorithm on the graph in Figure 34.

Definition 3.1.16 Consider a branching B in a directed graph $G = (V, E)$. An edge $e \in E \setminus B$ is called *eligible* if

$$(B \setminus \{(u, v) \in B : v = \text{head}(e)\}) \cup \{e\}$$

is also branching in G.

Lemma 3.1.17 *An edge $e = (s, t) \in E \setminus B$ is eligible if and only if there does not exist a directed (t, s)-path in B.*

Proof. Because other arcs going to head(e) are explicitly removed, the only obstacle for

$$(B \setminus \{(u, v) \in B : v = \text{head}(e)\}) \cup \{e\}$$

being a branching is that it contains a cycle. However, that happens if and only if it contains a directed (t, s)-path, which happens if and only if B contains one. □

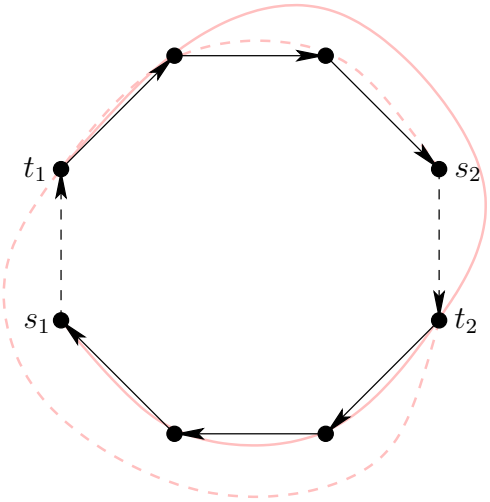

Figure 35: The cycle in the proof of Lemma 3.1.18.

Lemma 3.1.18 *Let B be a branching in a directed graph $G = (V, E)$ and C a directed cycle in G. If no edge from $C \setminus B$ is eligible then $|C \setminus B| = 1$.*

Proof. We cannot have $|C \setminus B| = 0$, since then $C \subseteq B$ and B would not be a branching.

Now we prove that we cannot have $|C \setminus B| = 2$ with all edges non-eligible. In Figure 35 the situation with the cycle C has been drawn. We have drawn also the arcs from B which are in C. In this picture there are six such arcs. The two which we are assume are eligible are (s_1, t_1) and (s_2, t_2). By Lemma 3.1.17, there exists a (t_1, s_1)-path in B which, because of B being a branching, must follow the path from t_2 to s_1 in C. Similarly there is a (t_2, s_2)-path in B. But now inside B we find a path from t_2 to t_1 and one from t_1 to t_2. We conclude that B contains a directed cycle. This is a contradiction.

The proof that $|C \setminus B| \neq k$ for all $k > 2$ works the same way. We leave out the proof here. □

Proof of Theorem 3.1.10. Let B be an optimum branching. Assume that B contains as many edges from the maximal critical graph H as possible. By Lemma 3.1.18 it suffices to prove that no arc from $H \setminus B$ is eligible. If some e was eligible, then introducing it

$$(B \setminus \{(u, v) \in B : v = \text{head}(e)\}) \cup \{e\}$$

would give a branching with one more arc from H (because an arc which is removed cannot be in H because H is a critical graph already containing e). It would also be optimum because substituting e cannot lower the weight of the matching because it is critical. That we now have an optimum branching with more edges from H than B contradicts the assumption on B. We conclude that no edge is eligible as desired. □

We notice that the optimum branching of Theorem 3.1.10 can also be chosen to satisfy that if there is no arc from B entering a cycle C_i then $C_i \setminus B = \{e_i^0\}$.

We are now ready to prove Theorem 3.1.12.

Proof of Theorem 3.1.12. Let B be an optimum branching satisfying the condition of Theorem 3.1.10. We construct a branching B' in G' of weight

$$w'(B') = w(B) - \sum_{i=1}^{k} w(C_i) + \sum_{i=1}^{k} w(e_i^0)$$

proving that any optimum branching has at least this weight.

To construct B', simply restrict B to G' in the natural way. The effect of collapsing C_i is that the weight drops by (passing from w weight to w' weights):

- $w(C_i) - w(e_i^0)$ if there is no arc in B ending at C_i
- $w(C_i) - w(\tilde{e}) - (w(\tilde{e}) - w(e_i^0)) = w(C_i) - w(e_i^0)$ otherwise

Summing for all cycles, we get a drop by $\sum_{i=1}^{k} w(C_i) - \sum_{i=1}^{k} w(e_i^0)$ as desired.

Now let $\tilde{B}'$ be and optimum branching of G'. We wish to construct an optimum branching $\tilde{B}$. This is done as in Algorithm 3.1.14. The effect is now the opposite as above. We conclude that there is a branching of weight

$$w(\tilde{B}) = w'(\tilde{B}') + \sum_{i=1}^{k} w(C_i) - \sum_{i=1}^{k} w(e_i^0)$$

and that any optimum branching must have at least this weight.

By optimality, $w'(\tilde{B}') \geq w'(B')$ and $w(B) \geq w(\tilde{B})$. Combining our two inequalities and two equations we get the conclusion of the theorem

$$w(B) - w'(\tilde{B}') = \sum_{i=1}^{k} w(C_i) - \sum_{i=1}^{k} w(e_i^0)$$

for optimum branchings B and $\tilde{B}'$. □

Exercise 3.1.19 Consider a town without central water supply. We want to install water towers and pipes. The town has certain important sites $v_1, \ldots, v_n$ which we represent by vertices in a directed graph G. We want to connect all these sites to a water tower. Furthermore we suppose that we can install a water tower at any of these sites at the expense of K for each tower. An alternative to installing a water tower at a site is connecting it to another site with pipes. If a connection with water flowing from site u to site v is possible, then G will have the arc (u, v). The expense of installing a pipe taking water from u to v is given by a weighting $w : E \to \mathbb{R}$. We assume that the directed graph $G = (V, E)$ is connected. We want to determine an optimal way of connecting the sites with pipes or installing towers, so that the cost is minimised.

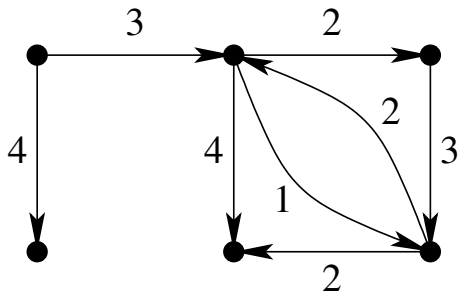

Figure 36: The map of the town in Exercise 3.1.19. The expense of installing a pipe along an arc is indicated. The expense of building a water tower is 3.

- Prove that the problem we want to solve is

  $$\min_{H \subseteq E}(w(H) + (|V| - |H|)K)$$

  where H is restricted to cycle-free subgraphs with at most one in-going arc for each vertex.

- Formulate this as an optimum branching problem.

- Find an optimal choice of locations for water towers in the graph of Figure 36.

- Can we also phrase the problem as an optimum branching problem if each tower has different cost to build?

Remark 3.1.20 The problem above becomes much harder when the towers are allowed to be located in between the specified sites. Look up *Steiner trees* for more information.

Exercise 3.1.21 Is the set of cycle free subgraphs of a directed graph (V, E) the independent sets of a matroid (with ground set E)?

Exercise 3.1.22 Can we phrase the optimum branching problem as a weighted matroid intersection problem for two matroids (Hint: be inspired by the Hamiltonian path problem formulated as an intersection of three matroids)?

4 Colourings

Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$. By a k-vertex *colouring* of G we mean a function $c : V \to C$ with C being a set of "colours" with $|C| = k$. We think of the function as assigning colours to the vertices of G. The book [4] provides a long list of open problems involving graph colourings.

A vertex colouring is called *proper* if for all $e = (u, v) \in E$ we have $c(u) \neq c(v)$. A graph is called k-vertex colourable if there exists a proper k-vertex colouring of G. The (vertex) chromatic number $\chi(G)$ is the smallest number k such that G is k-vertex colourable.

Similarly, we can colour edges. A k-edge colouring of a graph is a function $c : E \to C$ where C is a set of "colours" with $|C| = k$. We think of the function as assigning colours to the edges of G. An edge colouring is called *proper* if for every vertex v, the edges incident to v have different colours. Moreover, we require G to have no loops. A graph is called k-edge colourable if there exists a proper k-edge colouring G. The edge chromatic number $\chi'(G)$ is the smallest number k such that G is k-edge colourable.

Example 4.0.1 The graph G of Figure 30 has $\chi(G) = 3 = \chi'(G)$. Usually the vertex chromatic number and edge chromatic number are different.

4.1 Vertex colourings

Let $G = (V, E)$ be a graph. We define $\Delta = \max_{v \in V} d(v)$ to be the maximal degree of any vertex in G.

Theorem 4.1.1 *Every simple graph is $\Delta + 1$ colourable.*

Proof. Let $V = v_1, \ldots, v_n$. First assume that the vertices have no colours. We now iteratively assign colours to the vertices. For $i = 1, \ldots, n$, we consider the neighbours of v_i. There are at most Δ of them and therefore there is at least one colour left to colour v_i. Continuing like this we colour all vertices of the graph such that no two neighbours have the same colour. □

It is necessary that the graph is simple for the theorem to hold. If the graph contains a loop, then the vertex with the loop cannot be assigned a colour.

Example 4.1.2 We have $\chi(K_n) = \Delta + 1$ because for a complete graph $n = \Delta + 1$ and each vertex needs a different colour.

Example 4.1.3 If G is an odd cycle of length at least 3 then $\chi(G) = 3 = \Delta + 1$.

Remark 4.1.4 Consider a graph G with a proper vertex colouring. Let 1 and 2 be two colours. A restriction of G to the vertices of colours 1 and 2 can have several components. Let C be one of them. We can get a new proper colouring of G by exchanging the colours 1 and 2 in the component C. Such recolourings will be useful in the proof of the next theorem.

Theorem 4.1.5 (Brooks) *Let G be a connected simple graph. If G is not a cycle and not a complete graph then G is Δ-colourable.*

The following proof of Brooks' Theorem was originally given in [9] but also appears in [11].

Proof. First observe that the theorem is true for $\Delta = 0, 1, 2$.

Suppose now that the theorem did not hold in general. Then there would exist a graph G which is not a complete graph, not a cycle and not Δ-colourable, but after removing any vertex it will be one of these. Choose a vertex $v \in V$ to remove. We prove that $G' = G \setminus \{v\}$ is Δ-colourable. If $\Delta(G') < \Delta(G)$ this follows from Theorem 4.1.1. If $\Delta(G') = \Delta(G)$ we have $\Delta(G') \geq 3$ and G' is not a cycle. If G' was complete then G could not be connected.

If v had fewer than Δ neighbours in G then G' would also be Δ-colourable, which is a contradiction. Therefore v has exactly Δ neighbours $v_1, \ldots, v_\Delta$.

Observation 1: Any proper colouring of G' assigns different colours to $v_1, \ldots, v_\Delta$. If not, then G' could be Δ-coloured.

Let's now consider a particular Δ-colouring of G'. Define for $i \neq j$ the induced subgraphs B_{ij} of G' having only vertices with colours being those of v_i and v_j. Let C_{ij} be the component of Bij containing v_i.

Observation 2: Both v_i and v_j belong to C_{ij}. If v_j is not in C_{ij} then exchange colours of vertices in C_{ij} giving a new colouring of G' contradicting Observation 1.

We now argue that C_{ij} is a path between v_i and v_j. For this we must argue that the degree of v_i is 1 in C_{ij}. If it was ≥ 2, then the neighbours of v_i can have at most $\Delta - 1$ different colours. But now we can properly recolour v_i in the colour of v_j. This would contradict Observation 1. Clearly, since both v_1 and v_2 are in the connected component of C_{ij} the degree cannot be zero. Hence the degree of v_i is 1. The same holds for v_j. To prove that all other vertices in C_{ij} have degree 2, suppose not and let u be the first vertex starting from v_i where the degree was > 2. Then the neighbours of u have at most $\Delta - 2$ different colours. Δ-recolour u properly with a colour different from that of v_i. This new colouring would contradict Observation 2. Therefore C_{ij} is a path from v_i to v_j.

Observation 3: For i, j, k all different, C_{ij} and C_{ik} have only v_i in common. If they met at some other vertex u, this vertex must have the same colour as v_i. Of its at most Δ neighbours, 4 have the colours equal to v_j and v_k. Therefore it is possible to properly recolour u with a colour different from those of v_i, v_j and v_k. But in the new colouring v_i and v_j are not connected in the new C_{ij}, a contradiction to Observation 2.

Suppose now that G restricted to $v_1, \ldots, v_\Delta$ was complete, then also G restricted to $v, v_1, \ldots, v_\Delta$ would be complete. But then that would be all of

G since no vertex has degree higher than Δ and G is connected. But G was assumed not to be complete. Hence there exist non-adjacent v_i and v_j. Let k be different from i and j. We now have paths C_{ij} and C_{ik}. Consider the proper colouring where the colours along C_{ik} have been swapped. The path C'_{jk} of colour j, i goes along C_{ji} until the last vertex before v_i is reached. This last vertex u must have colour j and therefore the path C'_{ij} of colour j, k must involve u. However, now u is part of C'_{ij} and C'_{jk}, contradicting Observation 3.

We conclude that a minimal counter example does not exist. Therefore no counter example exists and the theorem is true. □

4.2 Chromatic polynomials

If we have a graph G, we may ask how many proper colourings it has. That of course depends on how many colours we have.

Example 4.2.1 The number of proper λ vertex colourings of the complete graph K_n is $\lambda(\lambda-1)\cdots(\lambda-n+1)$. An easy counting argument proves this. For the first vertex we have λ choices, for the next $\lambda-1$ and so on.

Example 4.2.2 Let G be the empty graph with n vertices and $\lambda \in \mathbb{N}$. Then G has exactly λ^n proper λ-vertex colourings, because for each vertex in G we can choose its colour freely.

Let $P(G, \lambda)$ denote the number of proper colourings of G with λ colours. We will prove that for any fixed graph G, the function $P(G, \lambda)$ is a polynomial function. If is called the *chromatic polynomial* of the graph.

For a graph G and an edge $e = (u, v) \notin E(G)$ we define:

- $G \cdot e$ to be the graph where u and v have been identified and parallel edges been replaced by a single edge.
- $G + e$ to be the simple graph G with e added.

Theorem 4.2.3 *Let G be a simple graph. Let $u, v \in V$ with $e = (u, v) \notin E$. Then*

$$P(G, \lambda) = P(G + e, \lambda) + P(G \cdot e, \lambda).$$

Proof. We simply observe:

- The set of proper λ-colouring of G with $c(u) = c(v)$ is in bijection with proper λ-colourings of $G \cdot e$.
- The set of proper λ-colouring of G with $c(u) \neq c(v)$ is in bijection with proper colourings of $G + e$.

□

Corollary 4.2.4 *Let $e = (u, v)$ be an edge in a simple graph G' then*

$$P(G', \lambda) = P(G' \setminus \{e\}, \lambda) - P(G' \cdot e, \lambda)$$

where $G' \cdot e$ means $(G' \setminus \{e\}) \cdot e$.

Proof. Simply apply the theorem to $G = G' \setminus \{e\}$. □

We now have a method to compute the function $P(G, \lambda)$ for a given graph G. We can either

- use the theorem repeatedly until we end up with complete graphs and apply the formula we already know, or
- use the corollary repeatedly until we end up with empty graphs and apply the formula we already know.

Example 4.2.5 In class we used the formula to compute the function for a 3-cycle with a single edge attached.

Theorem 4.2.6 *Let G be simple graph with n vertices. The function $P(G.\lambda)$ is the function of a polynomial of degree n with leading term λ^n and constant term 0. Moreover it has form*

$$\lambda^n - a_{n-1}\lambda^{n-1} + a_{n-2}\lambda^{n-2} - \cdots \pm 0$$

with $a_i \in \mathbb{N} = \{0, 1, 2, \dots\}$ meaning that coefficients have alternating sign, but some coefficients can be zero.

Proof. Induction on the number of edges.

Basis: For the empty graph (no edges) we have $P(G, \lambda) = \lambda^n$.

Step: Suppose the theorem is true for graphs with fewer edges. Pick $(u, v) \in E(G)$. Because a colouring of $G \setminus \{e\}$ induces a colouring on either G or $G \cdot e$ we have

$$P(G, \lambda) = P(G \setminus \{e\}, \lambda) - P(G \cdot e, \lambda).$$

By induction (because $|V(G \setminus \{e\})| = |V(G)|$ and $|V(G \cdot e)| = |V(G)| - 1$) we know

$$P(G \setminus \{e\}, \lambda) = \lambda^n - a_{n-1}\lambda^{n-1} + a_{n-2}\lambda^{n-2} \dots a_1\lambda^1$$
$$P(G \cdot e, \lambda) = \lambda^{n-1} - b_{n-2}\lambda^{n-2} + b_{n-3}\lambda^{n-3} \dots b_1\lambda^1$$

Subtracting the two expressions we get

$$P(G, \lambda) = \lambda^n - (a_{n-1} + 1)\lambda^{n-1} + (a_{n-2} + b_{n-2})\lambda n - 2 \cdots \pm (a_1 + b_1)\lambda$$

as desired. □

Exercise 4.2.7 What are the chromatic polynomials of

- K_4 with an edge removed?
- the graph being a disjoint union of two three-cycles: $\triangle$ $\triangle$?
- the graph being two three-cycles sharing a vertex: $\triangle\!\!\triangle$?
- the connected graph with 4 vertices and 4 edges containing a three cycle?

- the graph $\triangle\ \triangle$ but with a vertex from each triangle joined by an edge (7 edges in total)?
- the graph $\triangle\ \triangle$ but with two vertices from one triangle joined by edges to two vertices of the other (8 edges in total)?

Exercise 4.2.8 What is the colour chromatic number $\chi(G)$ of a graph G with chromatic polynomial

$$\lambda^6 - 8\lambda^5 + 26\lambda^4 - 43\lambda^3 + 36\lambda^2 - 12\lambda = \lambda(\lambda-1)(\lambda-2)^2(\lambda^2-3\lambda+3)?$$

4.3 Colourings of planar graphs

In this section we prove the 5-colouring Theorem 4.3.5.

Proposition 4.3.1 (Euler's formula) *Let $G = (V, E)$ be a connected plane graph with at least one vertex and with the complement of the graph in $\mathbb{R}^2$ having r regions. Then*

$$|V| - |E| + r = 2$$

Proof. Let H be a spanning tree of G. Because the complement of H has only one component and $|V(H)| = |E(H)| + 1$ for a tree, the formula holds for H.

Now we add in edges one at a time. Adding an edge will increase the number of regions by 1. After adding each edge, the formula still holds. At the end the formula holds for G. □

Recall the following basic theorem from the Graph Theory 1 course:

Theorem 4.3.2 *For a graph $G = (V, E)$ we have*

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

Corollary 4.3.3 *[1, Corollary 10.21] Let G be a connected simple planar graph with $|V| \geq 3$ then*

$$|E| \leq 3|V| - 6$$

Proof. Fix an embedding G of G and let G^* be the dual graph. Because G is simple, connected and with at least 3 vertices, $d(v) \geq 3$ for any vertex $v \in V(G^*)$. We have

$$2|E(G)| = 2|E(G^*)| = \sum_{v \in V(G^*)} d(v) \geq 3|V(G^*)| = 3(|E(G)| - |V(G)| + 2)$$

where the first equation follows from definition of dual graphs, the second from Theorem 4.3.2 and the last from Euler's formula. We conclude that

$$|E(G)| \leq 3|V(G)| - 6$$

as desired. □

From the three theorems above we deduce:

Corollary 4.3.4 *Every planar simple graph has a vertex of degree* ≤ 5.

Proof. We may assume that the graph is connected - if not just consider one of its components. If every vertex had degree ≥ 6, Theorem 4.3.2 and Corollary 4.3.3 give

$$6|V| \leq \sum_{v \in V} d(v) = 2|E| \leq 6|V| - 12,$$

which is a contradiction. □

We can now prove the 5-colouring theorem using a strategy similar to that of the proof of Brooks' Theorem 4.1.5. The proof follows the structure of the proof of [11, Theorem 9.12].

Theorem 4.3.5 (The 5-colouring Theorem) *Every simple planar graph* G *is* 5*-colourable.*

Proof. We do the proof by induction on the number of vertices. If G has no vertices, then it is 5-colourable.

For the induction step let G be a simple planar graph and consider a particular embedding of it in the plane. Pick $v \in V(G)$ with $d(v) \leq 5$. This can be done because of Corollary 4.3.4. Let $G' := G \setminus \{v\}$. By the induction hypothesis, it is possible to properly colour G' with the colours $\alpha_1, \ldots, \alpha_5$.

If $d(v) < 5$ or not all five colours were used to colour the neighbours of v in G, then we can properly colour G with five colours.

If $d(v) = 5$ and all neighbours of v have different colours, then assign names to the neighbours v_1, v_2, v_3, v_4, v_5 in such a way that the vertices are ordered clockwise around v. We may without loss of generality assume that the colour of v_i is α_i.

For $i \neq j$ define B_{ij} to be the subgraph induced by G' on the vertices with colours α_i and α_j. Let C_{ij} be the component of B_{ij} containing v_i.

If $v_j \in C_{ij}$ for all i, j, then there is a (v_1, v_3)-path inside C_{13}. This paths involves only colours α_1, α_3. There is also a (v_2, v_5)-path in C_{25}. Because of the "clockwise" assumption, the two paths must intersect. Because the graph is plane, this can only happen at vertices. However, the colours of vertices of C_{25} are α_2, α_5. This contradicts the colours of C_{13}.

We conclude that for some i, j we have $v_j \notin Cij$. We exchange the colours α_i and α_j in C_{ij}. This gives a new colouring of G', but where at most 4 colours are used to colour $v_1, \ldots, v_5$. We extend this to a colouring of G by colouring v with the colour α_i. Hence G is 5-colourable. □

The theorem can be made stronger.

Theorem 4.3.6 (The 4-colouring Theorem) *Every simple planar graph* G *is* 4*-colourable.*

Remark 4.3.7 The proof of the 4-colouring theorem is complicated and relies on checking 1482 special cases. This was done on a computer by Appel and Haken in 1977. At the time this was a controversial proof technique and it was unclear whether this should count as a mathematical proof.

Exercise 4.3.8 Are computer proofs controversial today?

An interpretation of the 4-colouring theorem is that any map (with connected countries) can be coloured with only 4 colours. Indeed consider the plane graph of the boarders. Its dual graph is also plane and has a vertex for each country. That a colouring of the countries is proper in this graph, now means that any two neighbouring countries have different colour.

4.3.1 A failing approach to the 4-colouring Theorem

The following conjecture is false:

Conjecture 4.3.9 (Tait, 1884) *Every 3-connected planar cubic graph G has a Hamiltonian cycle.*

If the conjecture was true, it would be easy to prove the 4-Colouring Theorem:

Let G be a planar graph and consider a particular embedding (Figure 37, first picture). Without loss of generality we assume that G is connected. We assume that the embedding is done with straight lines (Remark 2.1.3 – why are parallel edges not a problem?). We add edges to G to a obtain a plane graph G' where all the faces are triangles. We now add three new vertices u_1, u_2, u_3 to the vertex set so that the convex hull of these contains G' in its interior. We now further add edges to G' to obtain a graph F with all faces being triangles - also the outer one (Figure 37, second picture). To show that G has a proper 4-colouring, it suffices to show that the larger graph F has a proper 4-colouring.

Because F is a plane graph it has a well-defined dual graph F^*. Because the faces of F are triangles, the dual graph F^* is cubic. We can assume that it is also 3-connected (Exercise 4.3.12). Tait's conjecture gives us that F^* has a Hamiltonian cycle H (Figure 37, third picture). The embedding of H divides the plane into an inner and outer region – thereby dividing the vertex set of F into two non-empty sets V_{inner} and V_{outer} (Figure 37, fourth picture). Consider $F[V_{\text{inner}}]$ and $F[V_{\text{outer}}]$. These graphs are cycle-free: if $F[V_{\text{inner}}]$ contained a cycle C, that cycle would surround a triangular face of F whose dual vertex would be connected by the Hammilton cycle which will leave C crossing an edge whose ends consequently cannot both be in V_{inner} contradicting that $C \subseteq F[V_{\text{inner}}]$. A similar argument works for $F[V_{\text{outer}}]$. Cycle free graphs are trees and therefore have proper 2-colourings. Using two disjoint pairs of colours for $F[V_{\text{inner}}]$ and $F[V_{\text{outer}}]$ we get a proper 4-colouring of F. This is also a proper 4-colouring of the subgraph G, supposedly proving the 4-colour Theorem.

Exercise 4.3.10

- What does it mean to be cubic?

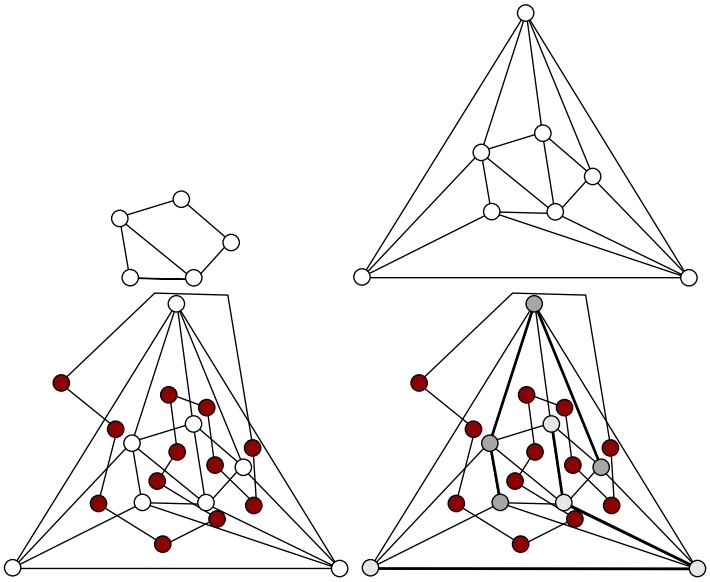

Figure 37: Various steps in the Tait "proof". All that remains in the last picture is to properly colour the light gray and dark gray vertices with two pais of different colours.

- What does it mean to be 3-connected?
- Find a planar cubic graph G that is not 3-connected and does not contain a Hamiltonian cycle.

Exercise 4.3.11 Why are parallel edges not a problem in the proof above?

Exercise 4.3.12 Is it really true that the graph F^* mentioned above is 3-connected? How do we fill the gap?

Unfortunately Tait's conjecture is false. This was proved by Tutte in 1946. We go through the proof in the following exercise.

Exercise 4.3.13 Consider the graph G in Figure 38. We wish to prove that there can be no Hamiltonian path from vertex u to vertex v in the graph. Choose according to your ZIP code: Prove that there is no Hamiltonian path from u to v...

8000-8099: including both edge a and c.

8100-8199: including both edge a and d.

8200-8299: including both edge b and c.

8300-8399: including both edge b and d.

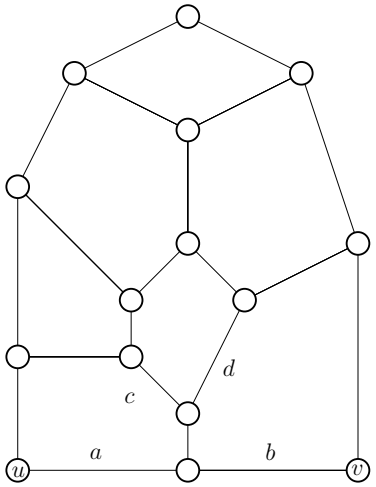

Figure 38: A Tutte fragment

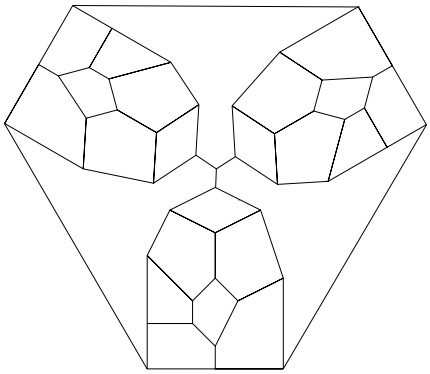

Figure 39: Three Tutte fragments forming a cubic graph without a Hamiltonian cycle

Conclude that there is no Hamiltonian path in G starting at u and ending at v. The graph G is called a Tutte fragment. Now we compose 3 such graphs to obtain the graph F in Figure 39. Argue that F does not have a Hamiltonian cycle, thereby proving that Tait's conjecture is false.

4.4 An open colouring problem

For every subset $V \subseteq \mathbb{R}^2$ we may define its *unit distance graph* $U_V := (V, E)$ where

$$E = \{(u, v) \in \mathbb{R}^2 \times \mathbb{R}^2 : d(u, v) = 1\}$$

Here $d(u, v)$ denotes the usual euclidean distance between the points u and v in the plane.

Example 4.4.1 A simple example would be to take $V = \{(0, 0), (1, 0)\}$, getting

a the complete graph on 2 vertices.

Example 4.4.2 The unit distance graph of

$$V = \{(\frac{1}{2}, \frac{1}{2}), (-\frac{1}{2}, \frac{1}{2}), (\frac{1}{2}, -\frac{1}{2}), (-\frac{1}{2}, -\frac{1}{2})\}$$

is a 4-cycle.

Exercise 4.4.3 What is the unit distance graph of

$$V = \{(\frac{1}{2}, 0), (-\frac{1}{2}, 0), (0, \frac{\sqrt{3}}{2}), (0, -\frac{\sqrt{3}}{2})\}?$$

Exercise 4.4.4 Prove that for any $n > 2$, the n-cycle C_n is a unit distance graph of some $V \subseteq \mathbb{R}^2$.

Exercise 4.4.5 Prove that the Petersen graph is a unit distance graph U_V for a suitable choice of $V \subseteq \mathbb{R}^2$.

In this section we will investigate a particular unit distance graph, namely the infinite unit distance graph of $V = \mathbb{R}^2$. This graph has an infinite number of vertices and edges and every vertex has infinite degree.

The Hadwiger-Nelson problem [4] is to answer the following question:

- What is the chromatic number of $U_{\mathbb{R}^2}$?

The answer to this question is unknown, but it is either 5, 6 or 7. We will not be able to prove this, but we will prove that the answer is either 4, 5, 6 or 7.

(Note that $U_{\mathbb{R}^2}$ is not necessarily a planar graph – if it was the chromatic number would be at most 4 by Theorem 4.3.6).

Exercise 4.4.6

- Find a drawing of the 10-vertex, 18-edge Golomb graph G on the internet.
- Why is G a unit distance graph?
- Find a proper 4-colouring of G.
- Prove that G has no proper 3-colouring.
- Prove that the chromatic number of $U_{\mathbb{R}^2}$ is at least 4.

Exercise 4.4.7

- Can we use that $\det((2,1),(-1,3)) = 7$ to argue that the parallellogram $\text{conv}((0,0),(2,1),(-1,3),(1,4))$ has has 6 interior lattice points?
- Use $f : \mathbb{Z}^2 \to \mathbb{Z}$, $f(x,y) = x + 5y$ to construct a 7-colouring of the graph $(\mathbb{Z}^2, E)$ with the property that the distance (measured in edges) between any two vertices with equal colour is at least 3, where

$$E = \{(u,v) \in \mathbb{Z}^2 \times \mathbb{Z}^2 : u - v \in \{\pm e_1, \pm e_2, \pm(e_1 - e_2)\}\}.$$

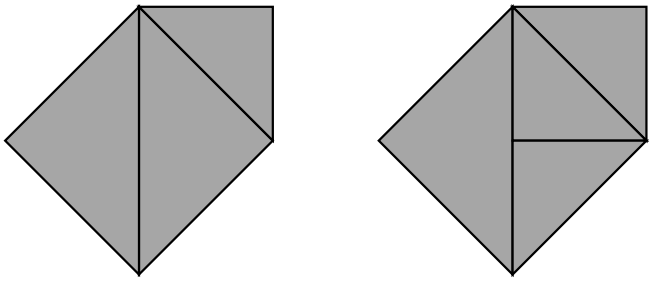

Figure 40: A collection of simplices forming a triangulation and a collection not forming a triangulation. See Example 4.5.1.

- Use regular hexagonal tiles with side length $\frac{1}{2} - \varepsilon$ to tile $\mathbb{R}^2$ (with $\varepsilon > 0$ being a small number), so that the ends of any edge of $U_{\mathbb{R}^2}$ must belong to different tiles.
- Inspired by the earlier vertex colouring, assign a colour to each of the tiles, so that no two tiles of the same colour have a common neighbouring tile.
- Prove that the colouring is in fact a proper 7-colouring of $U_{\mathbb{R}^2}$.

From the above exercises we conclude that the chromatic number of $U_{\mathbb{R}^2}$ is either $4, 5, 6$ or 7.

Exercise 4.4.8 Who is Aubrey de Grey?

4.5 Sperner's Lemma

A *d-simplex* in $\mathbb{R}^n$ is the convex hull of $d+1$ affinely independent vectors. Let $P \subseteq \mathbb{R}^n$ be a convex polytope. A *triangulation* of P is a finite collection Δ of simplices in $\mathbb{R}^n$ such that

- $P = \bigcup_{S \in \Delta} S$ and
- $S_1 \cap S_2$ is face of S_1 (and of S_2) for any $S_1, S_2 \in \Delta$.

Example 4.5.1 The collection of the 3 simplices in Figure 40 (left) is a triangulation, while the collection of the 4 simplices to the right is not.

If we take a polytope P and a collection of finite collection of points $V \in P$ including the vertices of P, it is possible to find a triangulation of P where the union of the set of vertices of the simplices equals V. (How?)

The *graph* of such triangulation Δ is (C, E) where $(u, v) \in E$ iff the convex hull $\text{conv}(u, v)$ is an edge of one of the simplices in Δ.

Sperner's Lemma is a statement about the colourings of such graphs. Let P be the $d-1$-dimensional simplex $\text{conv}(e_1, \ldots, e_d)$. Let $c : V \to \{1, \ldots, d\}$ be a colouring. We say that a simplex is *full-coloured* if all of its vertices have different colours.

Lemma 4.5.2 (Sperner's Lemma) *Consider the graph G of a triangulation Δ of $\mathrm{conv}(e_1, \ldots, e_d)$. If $c : V(G) \to \{1, \ldots, d\}$ is a colouring with $c(v) \neq i$ whenever $v_i = 0$, then Δ has a full-coloured $(d-1)$-simplex.*

Example 4.5.3 No matter how we complete the 3-colouring of the graph in Figure 41, it will contain a full-coloured simplex.

Proof. We will prove the lemma only for $d \leq 3$ and leave the general induction step as an exercise.

If $d = 1$, then the simplex is just a point and it is full-coloured (with a single colour).

If $d = 2$ we are colouring a path with two colours and with the two ends having opposite colours. By counting the number of times the colour changes as we go from one end to the other, we observe that it must have an odd number of full-coloured edges. So, indeed, the path has a full-coloured edge.

When $d = 3$ we notice that the following equation holds by counting the number of edges in the graph (with interior edges counted twice) in two different ways:

$$2 \cdot \#\mathrm{FullColouredEdges} - \#\mathrm{FullColouredBorderEdge} =$$

$$3 \cdot \#\mathrm{FullColouredSimplices} + 2 \cdot \#\mathrm{SemiColouredSimplices}$$

Here a simplex is semi-coloured if it has $d-1$ colours. Because the number of full-coloured border edges is odd for each edge of Δ, the left quantity is odd. We conclude that the right hand side of the equation is also odd, and therefore that the number of full-coloured simplices is odd. In particular it is ≥ 1. □

Exercise 4.5.4 Generalise the proof to any dimension.

Sperner used his lemma to give a surprisingly easy proof to a surprising theorem in Topology.

Waiting at the laundrymat you may close your eyes and let your mind remember the image of your clothes. As your eyes open again you will notice

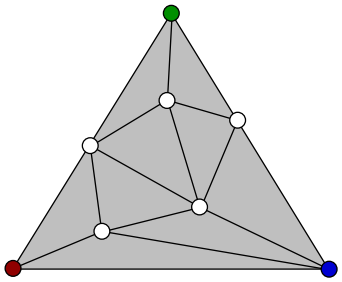

Figure 41: A partial 3-colouring of a graph of a triangulation of $\mathrm{conv}(e_1, e_2, e_3)$. It is impossible to complete the 3-colouring without making a fully coloured triangle. See Example 4.5.3.

that there (almost?) always is at least one point on a piece of clothes that did not move while you were not watching.

A similar situation appears if you take 2 sheets of A4 paper. First the sheets are lying on top of each other. Take the upper piece and fold it a few times along random lines, and place it back on the first sheet. If the folded sheet does not stick out from the lower piece, then in fact there is one point on your folded sheet that is now back to its original position.

This is the content of Brouwer's Fixed-point Theorem:

Theorem 4.5.5 (Brouwer's Fixed-point Theorem) *Let B denote the unit ball $\{x \in \mathbb{R}^n | \sum_i x_i^2 \leq 1\}$ and $f : B \to B$ be a continuous function. Then there exists a point $y \in B$ such that $f(y) = y$.*

Proof. We will prove a variant of the theorem where B is not the ball but a $d-1$-dimensional simplex $S_0 \subseteq \mathbb{R}^d$. Let $L > 0$ be a positive number. We choose a triangulation Δ of S such that every edge of a simplex in the triangulation is shorter than L. We make a colouring c with the following property for any v vertex in the graph of the triangulation:

$$f(v)_{c(v)} \leq v_{c(v)}$$

and further require $c(e_i) = i$ for $i = 1, \ldots, d$. It is possible to satisfy the first condition because if $f(v)_i > v_i$ for all i, then $\sum_i f(v)_i > \sum_i v_i = 1$, contradicting that v is in the simplex S_0. The second is easily satisfied.

By Sperner's Lemma, there exists a simplex $S_1 = \text{conv}(v_1^1, \ldots, v_d^1) \in \Delta$ that is fully coloured. If we now make a smaller choice of L we can subdivide S_1 to get a triangulation with edge lengths $< L$. We can colour the new vertices so that they satisfy the condition above. We apply Sperner's Lemma and get a simplex $S_2 = \text{conv}(v_1^2, \ldots, v_d^2)$.

Repeating this process, we get d sequences of points - one for each colour. Observe:

- The coordinate of each sequence converges because the condition on the edge length forces each such sequence to be Cauchy as L approaches 0.
- The rth's sequence satisfies $f(v_r^i)_r \leq (v_r^i)_r$ by the choice of colouring and therefore by continuity of f the limit u_r satisfies $f(u_r)_r \leq (u_r)_r$.
- The limit points of the each r sequences must be the same point u since the distances between the elements in the sequence approaches 0 as L approaches 0.

Now $u \in S_0$ because S_0 is closed. Moreover $f(u)_r \leq u_r$ on each coordinate. Because $\sum_i f(u)_i = 1$ and $\sum_i u_1 = 1$, we must in fact have $f(u)_r = u_r$ on each coordinate. This proves that u is the desired fixed-point. □

Exercise 4.5.6 Why is the laundrymat analogy not good? Which conditions are not satisfied?

Exercise 4.5.7 Construct a continuous map from the n-dimensional simplex in $\mathbb{R}^{n+1}$ to the unit ball $B \subseteq \mathbb{R}^n$ with a continuous inverse. Use this to prove the fixed-point theorem for unit balls.

Exercise 4.5.8 Since both Brouwer's Fixed-point Theorem and Jordan's Curve Theorem are topological statements, one might think that we could fix the gap in these notes by providing a proof of Theorem 2.1.5 using the Fixed-point theorem. Indeed this is the case see [8] for a proof. Read the proof. What other theorems from analysis are needed?

A English-Danish Dictionary

graph (graf) We distinguish between graphs and directed graphs. A graph is a pair (V, E) of vertices and edges. For this course the set of vertices V and the set of edges E are finite and the set E is a set of unordered pairs (u, v) where $u, v \in V$ may or may not be the same vertex. (In particular, we disallow parallel edges in this course.)

directed graph (orienteret graf) In a directed graph (V, E) the set E consists of ordered pairs (u, v) of vertices $u, v \in V$ called arcs.

vertex (knude)

edge (kant) An edge has two ends.

end of edge

arc (pil) Appear in directed graphs. An arc is an ordered pair (u, v) of vertices. The first vertex u is called the tail and the second v the head of the arc.

subgraph (delgraf) A graph $G' = (U', V')$ is a subgraph of $G = (U, V)$ if $U' \subseteq U$ and $E' \subseteq V$. We write $G' \subseteq G$.

spanning subgraph (udspændende delgraf) A sugraph $H \subseteq G$ is called spanning if $V(H) = V(G)$.

matching parring

vertex-induced subgraph For a graph $G = (V, E)$ and a subset $W \subseteq V$ we let $G[W]$ denote the graph with vertex set W and edge set being all edges of E having both ends in W.

edge-induced subgraph [1, page 50] For a graph $G = (V, E)$ and a subset $M \subseteq E$ we let $G[M]$ denote the graph with edge set M and vertex set being all the set of all ends of edges in M.

path (sti) An path is a graph $P = (V, E)$ where there exists names of the vertices $V = \{v_1, \ldots, v_n\}$ such that $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_n)\}$. The vertices v_1 and v_n are called the *ends* of P.

cycle (kreds) An n-cycle $C_n = (V, E)$ is a graph where there exists names of the vertices $V = \{v_1, \ldots, v_n\}$ such that $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_n), (v_n, v_1)\}$.

Forest (skov) A forest F is a graph without cyclces. (Meaning that no subgraph of F is a cycle.)

Tree (træ) A tree is a connected graph without cycle. (That is, it is a connected forest.)

loop (sløjfe) A loop is an edge with both ends being the same vertex.

bipartite 2-delt

component of connected component.

walk

spanning tree (udspændende træ) A spanning tree T of a graph G is a spanning subgraph of G that is also tree.

There is not necessarily one standard translation of each word. See `http://www.math.ku.dk/kurser/oa/Grafordbog.html` for alternatives.

B Exam topics (for fall semester 2018)

At the exam you will be assigned one of the 8 topics below at random. You will then present the topic for approximately 15-18 minutes. Because the topics vary in size it is important that you pick what to present. It is always good to present: a definition, a theorem, an example and (parts of) a proof. After (or during) your presentation we will ask questions about your presentation, your topic or another topic from class. The whole examination including evaluation takes 30 minutes. (Recall that there is no preparation time for this examination.)

1. **Matchings in bipartite graphs** Choose content from Sections (1.1,) 1.2, 1.3.
2. **Matchings in general graphs** Choose content from Section 1.4 and possibly from earlier sections.
3. **Maximal weight matching in general graphs** Section 1.7.
4. **Matroids and matroid polytopes** Section 2.7 and something from Sections 2.1-2.6.
5. **Matroids and the greedy algorithm** Section 2.8 and something from Sections 2.1-2.7.
6. **Optimum branching** Section 3.1 and possibly explain the connection to matroids.
7. **Chromatic polynomials**
8. **The 5-colouring theorem**

References

[1] John-Adrian Bondy and U. S. R. Murty. *Graph theory*. Graduate texts in mathematics. Springer, 2007.

[2] I. M. Gelfand, R. M. Goresky, R. D. MacPherson, and V. V. Serganova. Combinatorial geometries, convex polyhedra, and schubert cells, 1987.

[3] Martin Groetschel, Laszlo Lovasz, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Algorithms and combinatorics, 0937-5511. Springer, 1981.

[4] T. R. Jensen and B. Toft. *Graph Coloring Problems*. John Wiley & Sons, New York, NY, USA, 1994.

[5] Richard M. Karp. A simple derivation of Edmonds' algorithm for optimum branchings. *Networks*, 1(3):265–272, 1971.

[6] Niels Laurtizen. *Undergraduate Convexity*. World Scientific, 2013.

[7] Eugene Lawler. *Combinatorial Optimization: Networks and Matroids*. Dover Publications, 1976.

[8] Ryuji Maehara. The Jordan curve theorem via the Brouwer fixed point theorem. 91(10):641–643, 1984.

[9] L. S. Melnikov and V.G. Vizing. New proof of Brooks' theorem. *J. Combinatorial Theory*, 7:289–290, 1969.

[10] Gyula Pap. A matroid intersection algorithm. Technical Report TR-2008-10, Egerváry Research Group, Budapest, 2008. `www.cs.elte.hu/egres`.

[11] K. Thulasiraman and N.S. Swamy. *Graphs: Theory and Algorithms*. A Wiley interscience publication. Wiley, 1992.