Combinatorial optimisation for telecommunications
Lecture notes

MASCOTTE CNRS-INRIA-UNSA

January 23, 2012

Contents

Part I

Prerequisite knowledge

Chapter 1

Basic concepts

All the definitions given in this section are mostly standard and may be found in several books on graph theory like [1, 2, 3].

1.1 Graphs

A *graph* G is a pair (V,E) of sets satisfying $E \subset [V]^2$, where $[V]^2$ denotes the set of all 2-element subsets of V. We also assume tacitly that $V \cap E = \emptyset$. The elements of V are the *vertices* of the graph G and the elements of E are its *edges*. The vertex set of a graph G is referred to as $V(G)$ and its edge set as $E(G)$. An edge $\{x,y\}$ is usually written as xy. A vertex v is *incident* with an edge e if $v \in e$. The two vertices incident with an edge are its *endvertices*. An edge is said to *join* or *link* its two endvertices. Note that in our definition of graphs, there is no *loops* (edges whose endvertices are equal) nor *multiple edges* (two edges with the same endvertices).

Sometimes we will need to allow multiple edges. So we need the notion of *multigraph* which generalises the one of graph. A *multigraph* G is a pair (V,E) where V is the vertex set and E is a collection of elements of $[V]^2$. In a multigraph G, we say that xy is an edge of *multiplicity* m if there are m edges with endvertices x and y. We write $\mu(x,y)$ for the multiplicity of xy, and write $\mu(G)$ for the maximum of the edges multiplicities in G.

The *complement* of a graph $G = (V,E)$ is the graph $\overline{G}$ with vertex set V and edge set $[V]^2 \setminus E$. A graph is *empty* if it has no edges. A graph is *complete* of for all pair of distinct vertices u,v, $\{u,v\}$ is an edge. The complete graph on n vertices is denoted K_n. Trivially, the complement of an empty graph is a complete graph.

A *subgraph* of a graph G is a graph H such that $V(H) \subset V(G)$ and $E(H) \subset E(G)$. Note that since H is a graph we have $E(H) \subset E(G) \cap [V(H)]^2$. If H contains all the edges of G between vertices of $V(H)$, that is $E(H) = E(G) \cap [V(H)]^2$, then H is the subgraph *induced* by $V(H) = S$. It is denoted $G\langle S\rangle$. The notion of *submultigraph* and *induced submultigraph* are defined similarly. If S is a set of vertices, we denote by $G - S$ the (multi)graph induced by $V(G) \setminus S$. For simplicity, we write $G - v$ rather than $G - \{v\}$. For a collection F of elements of $[V^2]$, we write $G \setminus F = (V(G), E(G) \setminus F)$ and $G \cup F = (V(G), E(G) \cup F)$. As above $G \setminus \{e\}$ and $G \cup \{e\}$ are abbreviated to $G \setminus e$ and $G \cup e$ respectively. If H is a subgraph of G, we say that G is a *supergraph* of H.

Let G be a multigraph. When two vertices are the endvertices of an edge, they are *adjacent* and are *neighbours*. The set of all neighbours of a vertex v in G is the *neighbourhood* of G and is denoted $N_G(v)$, or simply $N(v)$. The *degree* $d_G(v) = d(v)$ of a vertex is the number of edges to which it is incident. If G is a graph, then this is equal to the number of neighbours of v.

Proposition 1.1. *Let $G = (V,E)$ be a multigraph. Then*

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

Proof. By counting *inc* the number of edge-vertex incidence in G. On the one hand, every edge has exactly two endvertices, so $inc = 2|E|$. On the other hand, every vertex v is an ednvertex of $d(v)$ edges, so $inc = \sum_{v \in V} d(v)$. □

The *maximum degree* of G is $\Delta(G) = \max\{d_G(v) \mid v \in V(G)\}$. The *minimum degree* of G is $\delta(G) = \max\{d_G(v) \mid v \in V(G)\}$. If the graph G is clearly understood, we often write Δ and δ instead of $\Delta(G)$ and $\delta(G)$. A graph is *k-regular* if every vertex has degree k. The *average degree* of G is $Ad(G) = \frac{1}{|V(G)|}\sum_{v \in V(G)} d(v) = \frac{2|E(G)|}{|V(G)|}$. The *maximum average degree* of G is $Mad(G) = \max\{Ad(H) \mid H \text{ is a subgraph of } G\}$.

Let G be a graph. A *stable set* or *independent set* in G is a set of pairwise non-adjacent vertices. In other words, a set S is stable if $G\langle S\rangle$ is empty. The *stability number* of G, denoted $\alpha(G)$ is the maximum cardinality of a stable set in G. Conversely, a *clique* in G is a set of pairwise adjacent vertices. In other words, a set S is a clique if $G\langle S\rangle$ is a complete graph. The *clique number* of G, denoted $\omega(G)$ is the maximum cardinality of a stable set in G.

1.2 Digraphs

A *multidigraph* D is a pair $(V(D),E(D))$ of disjoint sets (of *vertices* and *arcs*) together with two maps $tail : E(D) \to V(D)$ and $head : E(D) \to V(D)$ assigning to every arc e a *tail*, $tail(e)$, and a *head*, $head(e)$. The tail and the head of an arc are its *endvertices*. An arc with tail u and head v is denoted by uv and is said to *leave* u and to *enter* v; we say that u *dominates* v and write $u \to v$; we also say that u and v are *adjacent*. Note that a directed multidigraph may have several arcs with same tail and same head. Such arcs are called *multiple arcs*. A *multidigraph* without multiple arcs is a *digraph*. It can be seen as a pair (V,E) with a $E \subset V^2$. An arc whose head and tail are equal is a *loop*. All the digraphs we will consider in this monograph have no loops.

The multigraph G *underlying* a multidigraph D is the multigraph obtained from D by replacing each arc by an edge. Note that the multigraph underlying a digraph may not be a graph: there are edges uv of multiplicity 2 whenever uv and vu are arcs of D. *Subdigraphs* and *submultidigraphs* are defined similarly to subgraphs and submultigraphs.

Let D be a multidigraph. If uv is an arc, we say that u is an *inneighbour* of v and that v is an *outneighbour* of u. The *outneighbourhood* of v in D, is the set $N_D^+(v) = N^+(v)$ of outneighbours of v in G. Similarly, the *inneighbourhood* of v in D, is the set $N_D^-(v) = N^-(v)$ of inneighbours of v in G. The *outdegree* of a vertex v is the number $d_D^+(v) = d^+(v)$ of arcs leaving v and the

indegree of v is the number $d_D^-(v) = d^-(v)$ of arcs entering v. Note that if D is a digraph then $d^+(v) = |N^+(v)|$ and $d^-(v) = |N^-(v)|$. The *degree* of a vertex v is $d(v) = d^-(v) + d^+(v)$. It corresponds to the degree of the vertex in the underlying multigraph.

Proposition 1.2. *Let $D = (V,E)$ be a digraph. Then*

$$\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = |E|.$$

The *maximum outdegree* of D is $\Delta^+(D) = \max\{d^+(v), v \in V(D)\}$, the *maximum indegree* of D is $\Delta^-(D) = \max\{d^-(v), v \in V(D)\}$, and the *maximum degree* of D is $\Delta(D) = \max\{d(v), v \in V(D)\}$. When D is clearly understood from the context, we often write Δ^+, Δ^- and Δ instead of $\Delta^+(D)$, $\Delta^-(D)$ and $\Delta(D)$ respectively.

The *converse* of the digraph $D = (V,E)$ is the digraph $\overline{D} = (V(D), \overline{E})$ where $\overline{E} = \{(v,u) \mid (u,v) \in E\}$. A digraph D is *symmetric* if $D = \overline{D}$.

An *orientation* of a graph G is a digraph D obtained by substituting each edge $\{x,y\}$ by exactly one of the two arcs (x,y) and (y,x). An *oriented graph* is an orientation of graph.

1.3 Walks, paths, cycles

Let G be a multigraph. A *walk* in G is a finite (non-empty) sequence $W = v_0 e_1 v_1 e_2 v_2 \dots e_k v_k$ alternating vertices and edges such that, for $1 \leq i \leq k$, v_{i-1} and v_i are the endvertices of e_i. The vertex v_0 is called *start* of W and v_k *terminus* of W. They both are *endvertices* of W. The vertices $v_i, 1 \leq i \leq k-1$ are the *internal vertices*. One says that W *links* v_0 to v_k and that W is a (v_0, v_k)-*walk*.

Let A and B be to set of vertices. An (A,B)-*path* is a path whose start is in A, whose end is in B and whose internal vertices are not in $A \cup B$. We usually abbreviate $(\{a\}, B)$-path to (a,B)-path, $(A, \{b\})$-path to (A,b)-path and $(\{a\}, \{b\})$-path to (a,b)-path.

If $W_1 = u_0 e_1 u_1 e_2 u_2 \dots e_p u_p$ and $W_2 = v_0 f_1 v_1 f_2 v_2 \dots f_q v_q$ are two walks such that $u_p = v_0$, the *concatenation* of W_1 and W_2 is the walk $u_0 e_1 u_1 e_2 u_2 \dots e_p u_p f_1 v_1 f_2 v_2 \dots f_q v_q$. The *concatenation* of k walks $W_1, \dots, W_k$ such that for all $1 \leq i \leq k-1$ the terminus of W_i is the start of W_{i+1} is then defined inductively as the concatenation of W_1 and the concatenation of $W_2, \dots, W_k$.

If G is a graph, then the walk W is entirely determined by the sequence of its vertices. Very often, we will then denote $W = (v_0, v_1, \dots, v_k)$. The *length* of W is k, which is its number of edges (with repetitions). A walk is said to be *even* (resp. *odd*) if its length is even (resp. odd).

A walk whose start and terminus are the same vertex is *closed*. A walk whose edges are all distinct is a *trail*. A closed trail is a *tour*. A walk whose vertices are all distinct is a *path* and a walk whose vertices are all distinct except the start and the terminus is a *cycle*. Observe that a path is necessarily a trail and a cycle is a tour.

A path may also be seen as a non-empty graph $P = (V,E)$ of the form $V = \{x_0, x_1, \dots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k\}$ where the vertices x_i are all distinct. Similarly, a cycle may be seen as a non-empty graph $C = (V,E)$ of the form $V = \{x_0, x_1, \dots, x_k\}$ and $E = \{x_0x_1, x_1x_2, \dots, x_{k-1}x_k, x_kx_0\}$ where the x_i are all distinct.

Proposition 1.3. *Let G be a multigraph.*

(i) There is a (u,v)-walk in G if and only if there is a (u,v)-path.

(ii) An edge is in a closed trail if and only if it is in a cycle.

(iii) There is an odd closed walk, if and only if there is an odd cycle.

Proof. (i) Let $P = v_0e_1v_1e_2v_2\dots e_kv_k$ be a shortest (u,v)-walk. Then P is a path. Indeed suppose for a contradiction that there exists $i < j$ such that $v_i = v_j$. Then $v_0e_1\dots e_iv_ie_{j+1}\dots e_kv_k$ is a (u,v)-walk shorter than P, a contradiction.

(ii) Let $C = ve_1v_1e_2v_2\dots e_kv$ be a shortest closed trail. Then C is a cycle. Indeed suppose for a contradiction that there exists $1 \leq i < j < k$ such that $v_i = v_j$. Then $v_0e_1\dots e_iv_ie_{j+1}\dots e_kv$ is a (u,v)-trail shorter than C, a contradiction.

(iii) Let $C = ve_1v_1e_2v_2\dots e_kv$ be a shortest odd closed walk. Then C is an odd cycle. Indeed suppose for a contradiction that there exists $1 \leq i < j < k$ such that $v_i = v_j$. Then $v_0e_1\dots e_iv_ie_{j+1}\dots e_kv$ and $v_ie_{i+1}\dots e_jv_j$ are shorter closed walks than C. But the sum of the lengths of these two walks is the length of C and so is odd. So, one of the lengths is odd, a contradiction.

□

The *distance* between two vertices u and v in a multigraph is the length of a shortest (u,v)-walk or $+\infty$ if such a walk does not exists. It is denoted $dist_G(u,v)$, or simply $dist(u,v)$ if G is clearly understood from the context. The proof of (i) in the above proposition shows that a shortest (u,v)-walk (if one exists) is a (u,v)-path.

The word "distance" is well chosen because *dist* is a distance in the mathematical sense, that is a binary relation which is *symmetric* (for all $u,v \in V(G)$, $dist(u,v) = dist(v,u)$) and which satisfies the *triangle inequality*: for all $u,v,w \in V(G)$, $dist(u,w) \leq dist(u,v) + dist(v,w)$. See Exercise 1.12.

In multidigraphs, a *directed walk* is a finite (non-empty) sequence $W = v_0e_1v_1e_2v_2\dots e_kv_k$ alternating vertices and arcs such that, for $1 \leq i \leq k$, v_{i-1} is the start and v_i the terminus of e_i. *Directed trail*, *directed tour*, *directed path* and *directed cycle* are then defined similarly to *trail*, *tour*, *path* and *cycle*. Clearly, Proposition 1.3 has its analog for digraphs. Its proof is left in Exercise 1.11.

Proposition 1.4. *Let D be a multidigraph.*

(i) There is a directed (u,v)-walk in D if and only if there is a directed (u,v)-path.

(ii) An edge is in a closed directed trail if and only if it is in a directed cycle.

(iii) There is an odd closed directed walk, if and only if there is an odd directed cycle.

The *distance* between two vertices in a multidigraph is defined analogously to the distance in a mutigraph. However, it is no more a distance in the mathematical sense because it is not symmetric. However it satisfies the triangle inequality. See Exercise 1.12.

1.4 Connectivity and trees

A graph G is *connected* if for any two vertices u, v, there exists a (u,v)-path in G.

Proposition 1.5. *Let G be a graph and x a vertex of G. The graph G is connected if and only if for any vertex u in G, there is a (u,x)-path.*

The *connected components* of a graph are its maximal connected subgraph.

A graph with no cycle is a *forest*. It is also said to be *acyclic*. A connected forest is a *tree*. The *leafs* of a tree T are the vertices of degree at most 1.

Proposition 1.6. *Let G be a graph. If $\delta(G) \geq 2$ then G has a cycle.*

Proof. Let $P = (v_1, v_2, \ldots, v_k)$ be a path of maximal length. Since v_1 has degree 2 it is adjacent to a vertex $w \neq v_2$. The vertex w is in P otherwise $(w, v_1, v_2, \ldots, v_k)$ would be a longer path than P. Thus $w = v_j$ for some $j > 2$ and so $(v_1, v_2, \ldots, v_j, v_1)$ is a cycle. □

Proposition 1.6 implies that every forest has at least one leaf. In fact, it implies that every forest has at least two leaves.

Corollary 1.7. *Every forest on at least two vertices has at least two leaves.*

Proof. By induction on the number of vertices, the result holding trivially for the two forests on two vertices.

Let F be a tree on n vertices, with $n \geq 3$. By Proposition 1.6, F has at least one leaf x. The graph $F - x$ is a forest on $n-1$ vertices. By the induction hypothesis, it has two leaves y_1 and y_2. One of these two vertices, say y, is not adjacent to x since $d(x) \leq 1$. Hence y is also a leaf of F. □

Corollary 1.8. *For every tree T we have $|E(T)| = |V(T)| - 1$.*

Proof. By induction on the number of vertices of T, the result holding trivially if T is the unique tree on one vertex (K_1).

Let T be a tree on at least two vertices. By Corollary 1.7, T has a leaf x. Since T is connected, x has degree at least one, so $d(x) = 1$. Thus, $|E(T-x)| = |E(T)| - 1$. By the induction hypothesis, $|E(T-x)| = |V(T-x)| - 1 = |V(T)| - 2$. Hence, $|E(T)| = |V(T)| - 1$. □

Proposition 1.9. *Let T be a graph. Then the following four statements are equivalent:*

(i) T is a tree;

(ii) for any two vertices u, v of T, there exists a unique (u,v)-path;

(iii) T is connected-minimal, i.e. T is connected and $T \setminus e$ is not connected for all $e \in E(T)$;

(iv) T is acyclic-maximal, i.e. T is acyclic but $T \cup xy$ has a cycle for any pair $\{x,y\}$ of non-adjacent vertices in T.

Proof. (i)⇒(ii): By the contrapositive. Suppose that there exist two distinct (u,v)-paths $P = (p_1, p_2, \ldots, p_k)$ and $Q = (q_1, q_2, \ldots, q_l)$. Let i be the smallest index such that $p_{i+1} \neq q_{i+1}$ and let j be the smallest integer greater than i such that $p_j \in \{q_{i+1}, q_{i+2}, \ldots, q_l\}$. Let j' be the index for which $q_{j'} = p_j$. Then $(p_i, p_{i+1}, \ldots, p_j, q_{j'-1}, q_{j'-2}, \ldots, q_i)$ is a cycle.

(ii)⇒(iii): If there exists a unique path between any two vertices, then T is connected. Let $e = xy$ be an edge. Then (x,y) is the unique (x,y)-path in T. Thus $T \setminus e$ contains no (x,y)-path and so T is not connected. Hence T is connected-minimal.

(iii)⇒(i): By the contrapositive. Suppose that T is not tree. If T is not connected then it is not connected-minimal. Thus we may assume that T is connected and so T contains a cycle C. Let e be an edge of C. Let us show that $T \setminus e$ is connected which implies that T is not connected-minimal. Let x and y be two vertices. Since T is connected there is an (x,y)-path P in T. If P does not contain e then it is also a path in $T - e$. If P contains e then replacing e by $C \setminus e$ in P, we obtain an (x,y)-walk in $T \setminus e$. By Proposition 1.3, there is an (x,y)-path in $T \setminus e$.

(i)⇒(iv): If T is a tree then it is acyclic. Let us show that it is acyclic-maximal. Let x and y be two non-adjacent vertices. Then in T there is an (x,y)-path P since T is connected. The concatenation of P and (y,x) is a cycle in $T \cup xy$.

(iv)⇒(i): By the contrapositive. Suppose that $T$is not a tree. If it is not acyclic then it is not acyclic-maximal. Thus we may assume that T is not connected. So there are two vertices x and y for which there is no (x,y)-path in T. Let us show that $T \cup xy$ is acyclic which implies that T is not acyclic-maximal. Indeed if there were a cycle C, then it must contain xy because T is acyclic. Then $C \setminus xy$ would be an (x,y)-path in T, a contradiction. □

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

Corollary 1.10. *A graph G is connected if and only if it has a spanning tree.*

Proof. By induction on the number of edges of G. If G is connected-minimal, then by Proposition 1.9, G is a tree and thus a spanning tree of itself. If G is not connected-minimal, then by definition there is an edge e such that $G \setminus e$ is connected. By the induction hypothesis, $G \setminus e$ has a spanning tree which is also a spanning tree of G. □

1.5 Strong connectivity and handle decomposition

A digraph is *strongly connected* or *strong* if for any two vertices u, v there is a directed (u,v)-path. Observe that swapping u and v implies that there is also a directed (v,u)-path. The *strongly connected components* of a digraph G are its maximum strongly connected subgraphs.

The following proposition follows easily from the definition.

Proposition 1.11. *Let D be a strongly connected digraph. Then every arc is in a directed cycle.*

Proof. Let uv be an arc. Since D is strongly connected then there is a directed (v,u)-path in D. Its concatenation with (u,v) is a directed cycle containing uv. □

Definition 1.12. The *union* of two digraphs D_1 and D_2 is the digraph $D_1 \cup D_2$ defined by $V(D_1 \cup D_2 = V(D_1) \cup V(D_2)$ and $E(D_1 \cup D_2) = E(D_1) \cup E(D_2)$.

Let D be a digraph and H be a subdigraph of D. A *H-handle* is a directed path or cycle (all vertices are distinct except possibly the two endvertices) such that its endvertices are in $V(H)$ and its internal vertices are in $V(D) \setminus V(H)$. A *handle decomposition* of D is a sequence $(C, P_1, \ldots, P_k)$ such that:

- $C = D_0$ is a directed cycle;
- for all $1 \leq i \leq k$, P_i is a D_{i-1}-handle and $D_i = D_{i-1} \cup P_i$;
- $D_k = D$.

The following proposition follows easily from the definitions.

Proposition 1.13. *Let H be a strongly connected subdigraph of D. For any H-handle P, then $H \cup P$ is strongly connected.*

Proof. Left in Exercise 1.26 □

Since every strongly connected digraph contains a directed cycle (Proposition 1.11), an easy induction immediately yields the following.

Corollary 1.14. *Every digraph admitting a handle decomposition is strongly connected.*

The converse is also true: every strongly connected digraph admits a handle decomposition. In addition, it has a handle decomposition starting at any directed cycle.

Theorem 1.15. *Let D be a strongly connected digraph and C a directed cycle in D. Then D has a handle decomposition $(C, P_1, \ldots, P_k)$.*

Proof. Let H be the subdigraph of D that admits a handle decomposition $(C, P_1, \ldots P_k)$ with the maximum number of arcs. Since every arc xy in $E(D) \setminus E(H)$ with both endvertices in $V(H)$ is a H-handle, H is an induced subdigraph of D. Assume for a contradiction that $H \neq D$. Then $V(H) \neq V(D)$. Since D is strongly connected, there is an arc vw with $v \in V(D)$ and $w \in V(D) \setminus V(H)$. Since D is strongly connected, D contains a (w, H)-path P. Then, (v, w, P) is a H-handle in D, contradicting the maximality of H. □

1.6 Eulerian graphs

A trail in a graph G is *eulerian G* if it goes exactly once through every edge of G. A graph is *eulerian* if it has an eulerian tour.

Theorem 1.16 (Euler 1736)**.** *A connected graph is eulerian if and only if all its vertices have even degree.*

Proof. The condition can easily seen to be necessary. Indeed if a vertex appears k times (or $k+1$ if it appears as start and terminus) of an eulerian tour, it is incident to exactly $2k$ edges in the tour and so it has degree $2k$.

Let us now show that the condition is sufficient. The proof follows the lines of the following algorithm.

Algorithm 1.1.

1. Initialise $W := v$ for an arbitrary vertex v.
2. If all the edges of G are in W then return W.
3. If not an edge is not in $W = v_0e_1v_1 \dots e_lv_l$,
4. If an edge incident to v_l, say $e = v_lv_{l+1}$ is not in W, then $W := v_0e_1v_1 \dots e_lv_lev_{l+1}$; go to 2.
5. If not all the edges incident to v_l are in W. Since there is an even number of them, $v_0 = v_l$. Then G has an edge $e \notin W$ incident to a vertex v_i in W, for it is connected. Let $e = v_iu$ be this edge.
   $W := v_ie_{i+1}v_{i+1} \dots e_lv_le_1v_1 \dots e_iv_ieu$; go to 2.

□

1.7 Exercises

Exercise 1.1. Show that K_n, the complete graph on n vertices, has $\binom{n}{2}$ edges.

Exercise 1.2. Build a cubic graph with 11 vertices. (*cubic:* $d(v) = 3$ for all vertex v.)

Exercise 1.3. Show that every graph has two vertices of same degree.

Exercise 1.4. Let G be a graph on at least 4 vertices such that for every vertex v, $G-v$ is regular. Show that G is either a complete graph or an empty graph.

Exercise 1.5. Let n and k be two integers such that $n > k$ and H be a graph on n vertices. Show that if $|E(H)| > (k-1)(n-k/2)$ then H has a subgraph of minimum degree at least k.

Exercise 1.6 (Jealous husbands)**.**
Three jealous husbands and their wives want to cross a river. But they just have a small boat in which at most two persons can fit. None of the husbands would allow his wife to be with another man unless he is present. Draw the graph of all the possible distributions across the river and advice the walkers on the method to cross the river.

Exercise 1.7 (Dog, goat, cabbage)**.**
A man wants to cross a river with his dog, his goat and his (huge) cabbage. Unfortunately, the

man can cross the river with at most one of them. Furthermore, for obvious reasons, the man cannot leave alone on one bank neither the goat and the dog nor the cabbage and the goat. Draw the bipartite graph of all permissible situations. How does the man do to cross the river?

Exercise 1.8. Let u and v be two vertices of a graph G. Show that, if u and v have odd degree and all the other vertices have even degree, then there is a (u,v)-path in G.

Exercise 1.9. Show that in a graph two paths of maximum length have a vertex in common.

Exercise 1.10. Find what is wrong in the following statement: *An edge is in a closed trail if and only if it is in a cycle.*

Exercise 1.11. Show Proposition 1.4.

Exercise 1.12. 1) Show that of G is a multigraph then $dist_G$ is symmetric and satisfies the triangle inequality.

2) Show that of D is a multidigraph then $dist_D$ satisfies the triangle inequality but may be non-symmetric.

Exercise 1.13. Let D be a digraph without directed cycles. Show that D has a vertex with indegree zero.

Exercise 1.14. Let $G=(V,E)$ be a graph. Show the following.
(1) If $|E|\geq|V|$ then G contains a cycle.
(2) If $|E|\geq|V|+4$ then G contains two edge-disjoint cycles.

Exercise 1.15. Let G be a graph of minimum degree at least 3. Show that G contains an even cycle.

Exercise 1.16. Let G be a connected graph. Show that there exists an orientation of G such that the outdegree of every vertex is even if and only if G has an even number of edges.

Exercise 1.17. Let G be a graph. Its *diameter* is the maximum distance between two vertices.

1) Show that if G has a diameter at least 3 then its complement $\overline{G}$ has diameter at most 3.

2) Deduce that every self-complementary graph ($G=\overline{G}$) has diameter at most 3.

3) For $k=1,2,3$, give an example of a self-complementary graph with de diameter k.

Exercise 1.18. Let T be a tree on at least two vertices. Show that if T has no vertex of degree 2 then T has at least $|V(T)|/2+1$ leaves.

Exercise 1.19 (Helly property for trees)**.** Let $T_1,\ldots,T_k$ be subtrees if a tree T. Show that if $T_i\cap T_j\neq\emptyset$ for all i,j, then $\bigcap_{i=1}^{k}T_i\neq\emptyset$.

Exercise 1.20. Is the complement of a non-connected graph always connected?
Is the complement of a connected graph always non-connected?

Exercise 1.21. 1) Show that every connected graph G has a vertex x such that $G-x$ is connected.
2) Does the same hold for strongly connected digraphs?

Exercise 1.22. Let G be a connected graph and e an edge of G. Show that G has a spanning tree containing e.

Exercise 1.23. A graph is *cherry-free* if every vertex has at most one neighbour of degree 1. Prove that a connected cherry-free graph has two adjacent vertices u and v such that $G-\{u,v\}$ is connected.
Hint: Consider a path of maximum length.

Exercise 1.24. Let G be a connected graph and $(V_1,V_2,\ldots,V_n)$ a partition of $V(G)$ such that $G\langle V_i\rangle$ is connected for all $1\leq i\leq n$. Show that there exists two indices i and j such that $G-V_i$ and $G-V_j$ are connected.

Exercise 1.25. The aim of this exercise is to prove that if a graph has n vertices, m edges and k connected components then $n-k\leq m\leq \frac{1}{2}(n-k)(n-k+1)$.
1) Let G be a graph on n vertices with m edges and k connected components.

a) Show that if G is connected then $m\geq n-1$.

b) Deduce that if G has k connected components then $m\geq n-k$.

2) Suppose now that G is a graph on n vertices and k connected components with the maximum number of edges.

a) Show that all the connected components of G are complete graphs.

b) Show that if G has (at least) two connected components then one of them has a unique vertex.

c) Deduce that G has $\frac{1}{2}(n-k)(n-k+1)$ edges.

Exercise 1.26. Prove Proposition 1.13.

Exercise 1.27. Let D be a strongly connected digraph and D' a strongly connected subdigraph of D. Show that any handle decomposition $(C,P_1,\ldots,P_k)$ of D' may be extended into a handle decomposition $(C,P_1,\ldots,P_k,\ldots,P_l)$ of D.

Exercise 1.28. Let D be a strongly connected digraph of minimum outdegree 2. Prove that there exists a vertex v such that $D-v$ is strongly connected.

Exercise 1.29. Show that a graph has an eulerian trail if and only if it has zero or two vertices of odd degree.

Exercise 1.30. Let $G=(V,E)$ be a graph such that every vertex has even degree and $|E|\equiv 0[3]$. Prove that E can be partitionned into $l=\frac{|E|}{3}$ sets $E_1,\ldots,E_l$ such that for all $1\leq i\leq l$, the graph induced by E_i is either a path of length 3 or cycle of length 3.

Bibliography

[1] J. A. Bondy and U. S. R. Murty. *Graph theory*, Series: Graduate Texts in Mathematics, Vol. 244, Springer, 2008.

[2] R. Diestel. *Graph Theory*, 3rd edition. Graduate Texts in Mathematics, 173. Springer-Verlag, Berlin and Heidelberg, 2005.

[3] D. B. West. *Introduction to graph theory*. Prentice Hall, Inc., Upper Saddle River, NJ, 1996.

Chapter 2

Searches in graphs and digraphs

2.1 Searches and connectivity in graphs

Finding the connected component of a vertex v in a graph is not difficult. For this purpose, we need a list of edges to be explored, initially containing all edges, and a list of already explored vertices, initially containing only v. At each step, a new edge ab is explored with a already explored and b is added in the list of explored vertices if it had not been explored yet. When the procedures stops, the set of explored vertices is the connected component of v. This algorithm may be modified to return a spanning tree T of the connected component of v.

Algorithm 2.1 (Search)**.**

1. Mark v and initialize L to the set of all edges incident to v; $V(T) := \{v\}$, $E(T) := \emptyset$.
2. If $L = \emptyset$, then return T; else let $ab \in L$. $L := L \setminus \{ab\}$.
3. If b is not marked, then mark it; $V(T) := V(T) \cup \{b\}$; $E(T) := E(T) \cup \{ab\}$; add all the edges incident to b to L.
4. Go to 2.

There are two well-known searches which correspond to two different orderings of the edges:

- the breadth-first search (BFS) (Algorithm 2.2) explores first the neighbours of v, then the neighbours of its children;
- the depth-first search (DFS) (Algorithm 2.3) explores first all the vertices of a branch pending in v.

The difference between these two approaches is that the vertices are stored either in a queue (FIFO) or in a stack (LIFO). A *queue* is just a list which is updated by either adding a new

element to one end (its *tail*) or removing an element from the other end (its *head*). A *stack* is simply a list, one end of which is identified as its *top*; it may be updated either by adding a new element as its top or else by removing its top element.

The following algorithms also compute the connected component C of v and a spanning tree of C.

Algorithm 2.2 (Breadth-First Search)**.**

1. Mark v, $V(T) := \{v\}$, $E(T) := \emptyset$ and initialize a queue Q to v.
2. If $F = \emptyset$ then return T. Else, remove the the vertex u fromt the head of Q.
3. For every unmarked vertex w adjacent to u, $V(T) := V(T) \cup \{w\}$; $E(T) = E(T) \cup \{uw\}$; add w to the tail of Q and mark w.
4. Go to 2.

Algorithm 2.3 (Depth-First Search)**.**

0. For every vertex, $L(u) := N(u)$.
1. Mark v; $V(T) := \{v\}$; $E(T) := \emptyset$ and initialize a stack P to v.
2. If $P = \emptyset$, then return T. Else, let u be the vertex on top of the stack P.
3. If $L(u) = \emptyset$, then remove u from the top of the stack P and go to 2.
4. Else, remove a vertex w from $L(u)$.
5. If w is marked go to 3. Else, $V(T) := V(T) \cup \{w\}$; $E(T) = E(T) \cup \{uw\}$; add w on top of P and mark w.
6. Go to 2.

A tree obtained by running a breadth-first search is called a *breadth-first search tree* or *BFS-tree*. Similarly, a tree obtained by running a depth-first search is called a *depth-first search tree* or *DFS-tree*. If the search is run from vertex v, this vertex is called the *root* of the search tree.

Observe that in Algorithms 2.1, 2.2 and 2.3 every edge is examined at most twice (once per endvertex). These algorithms can be modified in order to compute all the connected components of a graph so that every edge is examined at most twice. For this purpose, while some vertex does not belong to a connected component (i.e., has not been marked), it is sufficient to compute its connected component.

2.1.1 Distance in graphs

A nice property of a breadth-first search tree is that is can give the distance from the root r to all other vertices. Therefore we need at each vertex a value $l(u)$, called *level* of u, which corresponds to $dist_T(r,u)$ as well as $dist_G(r,u)$ as we will show later. Hence the following algorithm is the following:

Algorithm 2.4 (Distance from r)**.**

1. Mark r, $l(r) := 0$ and initialize a queue Q to v. [[$V(T) := \{v\}$; $E(T) := \emptyset$;]]
2. If $F = \emptyset$ then return l. Else, remove the first vertex u of Q.
3. For every unmarked vertex w adjacent to u, $l(w) := l(u) + 1$; add w to Q and mark w. [[$V(T) := V(T) \cup \{w\}$; $E(T) = E(T) \cup \{uw\}$;]]
4. Go to 2.

Observe that the construction of the tree T (operation between brackets at Step 1 and 3) is practically useless. It just help us to show that the function l has the properties we announced. The first ones justifies our referring to l as the level function.

Theorem 2.1. *Let r be a vertex of a connected graph G and T be a BFS-tree (as constructed by Algorithm 2.4). Then:*

(i) for every vertex v of G, $l(v) = dist_T(r,v)$;

(ii) every edge of G joins vertices on the same or consecutive levels of T; that is

$$|l(u) - l(v)| \leq 1, \quad \textit{for all } uv \in E(G).$$

Proof. The proof of (i) is left to the reader in Exercise 2.1. To establish (ii), it suffices to prove that if $uv \in E(G)$ and $l(u) < l(v)$, then $l(u) = l(v) - 1$.

We first establish, by induction on $l(u)$, that if u and v are any two vertices such that $l(u) < l(v)$, then u joined Q before v. This evident if $l(u) = 0$, because u is then the root r of T. Suppose that the assertion is true whenever $l(u) < k$, and consider the case $l(u) = k$, where $k > 0$. Let x be the predecessor of u, that is the vertex which is explored when we add u to Q. Then it follows from line 3 of Algorithm 2.4 that $l(x) = l(u) - 1$. Similarly, if y is the predecessor of v then $l(y) = l(v) - 1$. By induction, x joined Q before y. Therefore u being a neighbour of x, joined Q before v.

Now suppose that $uv \in E(G)$ and $l(u) < l(v)$. If u is the predecessor of v, then $l(u) = l(v) - 1$, again by line 3 of Algorithm 2.4. If not, let y be the predecessor of v. Because v was added to T by the edge yv, and not the edge uv, the vertex y joined Q before u, hence $l(y) \leq l(u)$ by the claim established above. Therefore $l(v) - 1 = ly) \leq l(u) \leq l(v) \leq l(v) - 1$, which implies $l(u) = l(v) - 1$. □

The following theorem shows that Algorithm 2.4 runs correctly.

Theorem 2.2. *Let T be a BFS-tree of a connected graph G, with root r. Then:*

$$dist_T(r,v) = dist_G(r,v), \quad \text{for all } v \in V(G).$$

Proof. Clearly, $dist_T(r,v) \geq dist_G(r,v)$ because T is a subgraph of G.

Let us establish the opposite inequality by induction on the length of a shortest (r,v)-path, the proposition holding trivially when the length is 0.

Let P be a shortest (r,v)-path in G, where $v \neq r$, and let u be the predecessor of v on P. The (r,u)-subpath of P is a shortest (r,u)-path, and $d_G(r,u) = d_G(r,v) - 1$. By induction, $l(u) \leq d_G(r,u)$, and by Theorem 2.1-(ii), $l(v) = l(u) \leq 1$. Therefore

$$dist_T(r,v) = l(v) \leq l(u) + 1 \leq d_G(r,u) + 1 = d_G(r,v).$$

□

2.2 Searches and strong connectivity in directed graphs

One can explore digraphs in much the same way as graphs, but by growing arborescences rather than (rooted trees). An *arboresence* is an orientation of a rooted tree in which all the arcs are directed from the root to the leaves. It can be seen as a digraph in which every vertex has indegree 1 except one called the root which has indegree 0. As with search in graph, search in digraph may be refined by restricting the choice of the arc to be added at each stage. In this way, we obtain directed versions of breadth-first search and depth-first search. We now discuss how search can be applied to find the strongly connected components of a digraph.

To test if a digraph $D = (V,E)$ is strongly connected, one has to check for every pair u,v of vertices if there is a (u,v)-dipath. Checking if such a path exists can be done by performing a search, so running $\binom{|V(D)|}{2}$ searches will do the job. However running a search from a vertex u finds all the vertices v that can be reached from u. So, in fact, one just need to run at most $|V|$ searches (one per vertex) yielding a total time $O(|V||E|)$.

The following proposition will yield an algorithm that test if a digraph is strong by running only two searches.

Proposition 2.3. *Let D be a digraph and v a vertex of D. D is strongly connected if and only if, for every $u \neq v$, there are a (u,v)-path and a (v,u)-path.*

Proof. If D is strongly connected, by definition, for every $u \neq v$, there are a (u,v)-path and a (v,u)-path.

Let us assume now that for every $u \neq v$, there are a (u,v)-path and a (v,u)-path. Let us show that D is strongly connected. Let u and w be two distinct vertices of D. There are a (u,v)-path P and a (v,w) path Q the concatenation of which is a (u,w)-walk. By Proposition 1.3, there is a (u,w)-path. □

We now describe an algorithm that computes the strongly connected components of a vertex v of a digraph. It is based on two searches starting from v, the first one in D and the reverse $\bar{D}$ of D. During the first search, the vertices u reachable from v are marked 1. During the second search the vertices u from which v can be reached marked 2 and included in the strongly connected component of v if they are already marked 1 (See Figure 2.1).

Algorithm 2.5 (Strongly connected component)**.**

1. Search D starting from v marking the vertices with 1.
2. Search $\bar{D}$ starting from v marking the vertices with 2.
3. Return the vertices marked with 1 and 2.

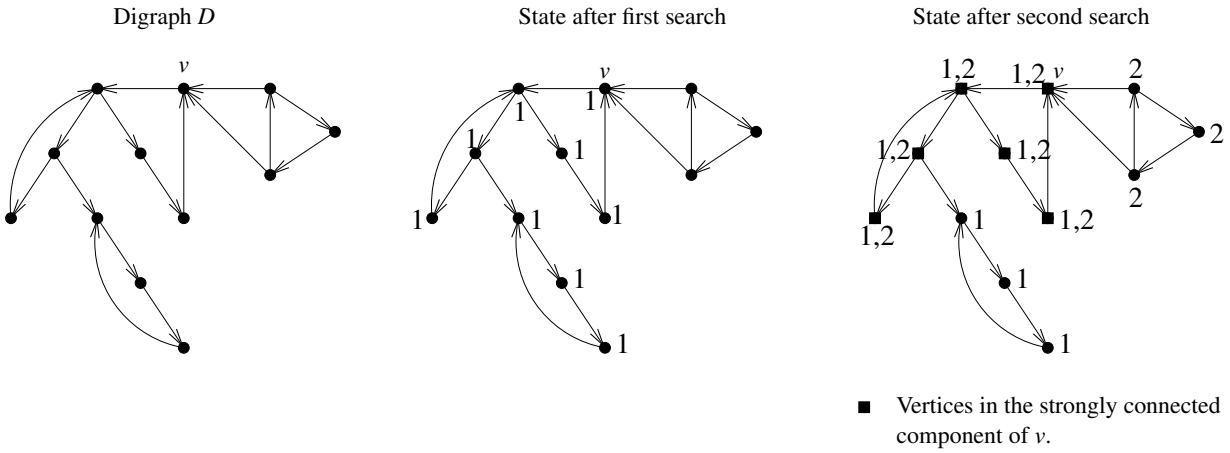


Figure 2.1: Execution of Algorithm 2.5

Contrary to Algorithm 2.1, Algoritm 2.5 does not give all strongly connected components of D in a single search examining each edges twice. Indeed, let us consider the digraph D with $V(D) = \{v_1, v_2, \ldots, v_n\}$ and$E(D) = \{(v_i, v_j) \mid i < j\}$. The components consist of each $\{v_i\}$. Hence, $|V|$ executions of Algoritm 2.5 must be done. Moreover, at each execution, all edges are considered.

2.2.1 Computing strongly connected components in one search

We now describe an algorithm that computes all strongly connected components of a digraph in time $O(|E|)$. It is a modified depth-first search in which two extra values are stored and updated. When a vertex is explored u it becomes *active* and is associated to two values $l(u)$ and $b(u)$. The first one $l(u)$, called *label* of u, corresponds to the order of appearance of u during the

search. It will never change. A vertex w such that $l(w) \geq l(u)$ is called a *successor* of u. A key ingredient of the algorithm is that as long as u is active, there is a directed (u,w)-path to each of its successors w. The second value $b(u)$ corresponds to the smallest label of a vertex reachable from u in the subdigraph induced by the explored arcs. Thus it needs to be updated when new arcs are explored.

Algorithm 2.6 (All Strongly Connected Components)**.**

0. Initialize i to 0.
1. If all vertices are marked, then terminate.
2. Else $i := i+1$.
3. Let u be an unmarked vertex. $l(u) := i$, $b(u) := i$, and u becomes active.
4. If at least one arc leaving u, say (u,v), is not marked, then do
   - 4.1 Mark (u,v).
   - 4.2 If v has already been explored and is active, then update $b(u)$: $b(u) := \min(b(u), b(v))$.
   - 4.3 Else v is a new vertex. v becomes active; $i := i+1$; $l(v) := i$; $b(v) := l(v)$; $u := v$.
   - 4.4 Go to 4.
5. Else, all arcs leaving u are marked, the exploration of u is over:
   - 5.1 If $b(u) = l(u)$ then all active successors of u induce a strongly connected component: return it and all its vertices become inactive; Go to 1.
   - 5.2 Else $b(u) < l(u)$. Let w be the vertex from which u has been explored. Update $b(w)$: $b(w) := \min(b(w), b(u))$; $u := w$; Go to 4.

Correctness of Algorithm 2.6: We will show by induction the following three points.

1) If $l(u) < l(v)$, and if u and v are active, then v is a successor of u;
2) At each step, for any active vertex v, there is a directed path from v to the vertex w with label $l(w) = b(v)$.
3) When the exploration of u terminates (Step 5), all the active vertices of $S(u) = \{v \text{ active } \mid b(u) \leq l(v) \leq l(u)\}$ are in the same strongly connected component as u.
4) $b(u) = l(u)$ if and only if $S(u)$ is a strongly connected component.

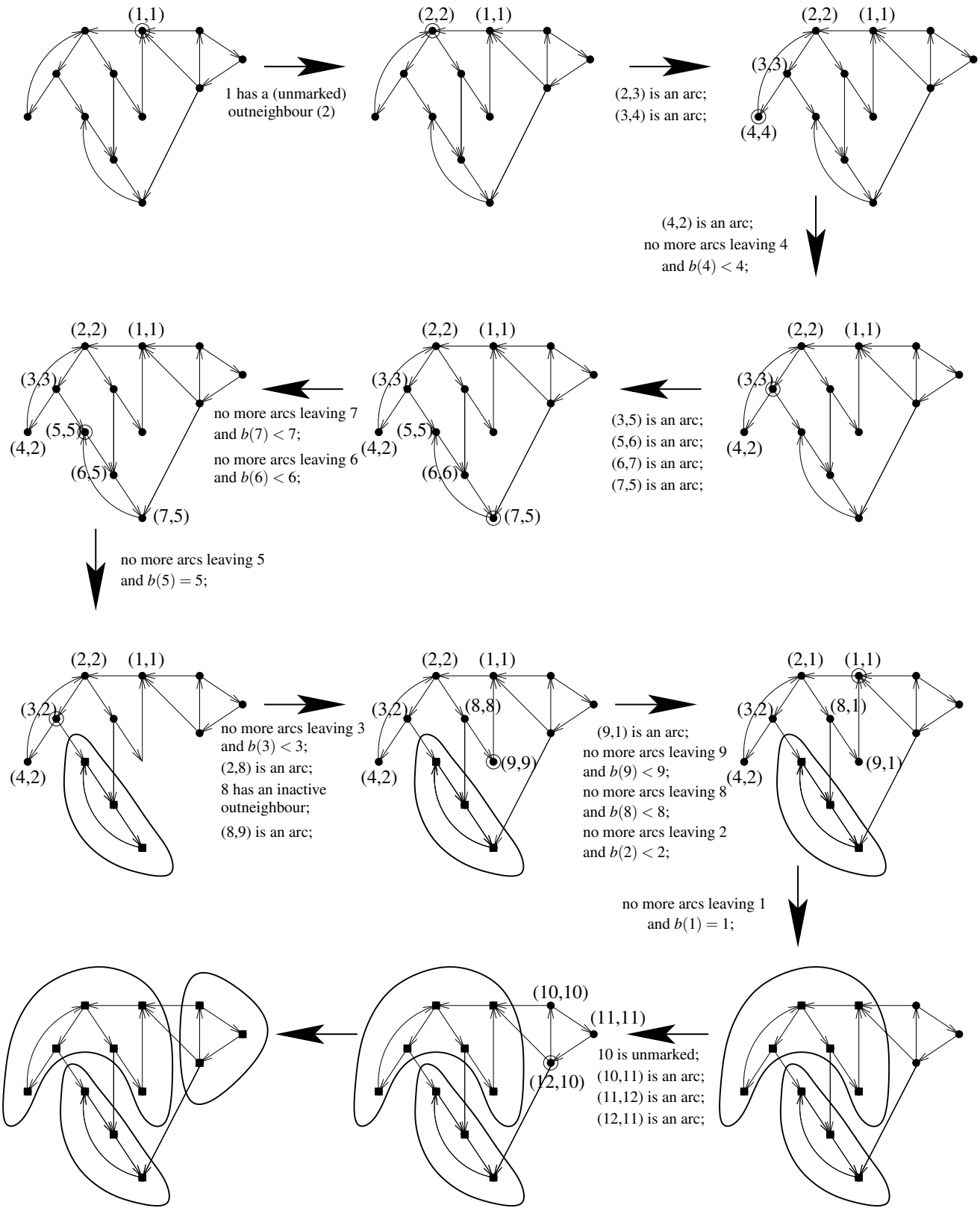


Figure 2.2: A run of Algorithm 2.6. The vertex in a circle is the current vertex u. At each step, $(l(v), b(v))$ is represented close to every active vertex v. Finally, once a vertex becomes inactive, it is depicted by a square.

1) and 2) Left to the reader.

3) From Proposition 2.3, it is sufficient to show that, for every v in $S(u)$, there is a directed (u,v)-path and a directed (v,u)-path. Let v be a vertex in $S(u)$.
Let us assume first that $l(v) < l(u)$. Then, by 1), there is a directed (v,u)-path. Let w be the vertex such that $l(w) = b(u)$. By 2), there is a directed (u,w)-path, and by 1) there is a (w,v)-path. The concatenation of these two paths is a directed (u,v)-walk. By Proposition 1.4 (i), there is a directed (u,v)-path.
Let us now assume that $l(v) > l(u)$. By 1) there is a (u,v)-path. Moreover, $b(v) < l(v)$, since otherwise, the vertex v would have become inactive at Step 5. Let v_1 be the vertex with label $l(v_1) = b(v)$. By 2), there is a directed (v,v_1)-path P_1. If $l(v_1) \leq l(u)$, then, by 1), there is a directed (v_1,u)-path whose concatenation with P_1 is a directed (v,u)-walk. Hence, a directed (v,u)-path exists according Proposition 1.3 (i). If $l(v_1) > l(u)$, then $b(v_1) < l(v_1)$, for otherwise, the vertex v_1 (and also v) would have become inactive at Step 5. Let v_2 be the vertex such that $l(v_2) = b(v_1)$. Using similar arguments, while $l(v_i) > l(u)$, we have $l(v_i) > b(v_i)$ and we denote by v_{i+1} the vertex such that $l(v_{i+1)} = b(v_i)$. Since the label of v_i strictly decreases, the sequence of the v_i's is finite. Let k be the last index of this sequence.Then $l(v_k) \leq l(u)$. By 2), for every $1 \leq i \leq k$, there is a directed (v_{i-1}, v_i)-path. Moreover, by 1), there is a directed (v_k,u)-path. The concatenation of all these paths is a directed (v,u)-walk, and then, by Proposition 1.4 (i), there is a directed (u,v)-path.

4) If $b(u) = l(u)$, then the arcs leaving $S(u)$ have their heads inactive (vertices in a distinct strongly connected component, by the induction hypothesis 3). Hence, $S(u)$ is a strongly connected component.
If $b(u) < l(u)$, then the vertex v labelled $b(u)$ is active. By 3), u and v are in the same strongly connected component but $v \notin S(u)$. Hence $S(u)$ is no a strongly connected component.

2.3 Bipartite graphs

A *bipartition* of a graph G is a partition (A,B) of $V(G)$ into two stable sets. Hence, every edge of G has an endvertex in A and the other in B. We often write $G = ((A,B),E)$ for a bipartite graph with bipartition (A,B).

Bipartite graphs satisfy some properties.

Proposition 2.4. *Let $G = ((A,B),E)$ be a bipartite graph.*

(i) for any two vertices u and v, all the (u,v)-walks have the same length parity.

(ii) if G is connected then it has only two bipartitions (A,B) and (B,A).

Proof. (i) Without loss of generality, we may assume that u is in A. Let $(v_0, v_1, \ldots, v_k)$ be a (u,v)-walk (so $v_0 = u$ and $v_k = v$). Since G is bipartite and $V_0 \in A$ then $v_1 \in B$ and so $v_2 \in A$. And so on by induction, if i is even then $v_i \in A$ and if i is odd $v_i \in B$. Thus if $v \in A$ then k is even and if $v \in B$ k is odd.

(ii) Let u be a vertex. Let A_0 (resp. A_1) be the set of vertices at even (resp. odd) distance to v_0 in G. By (i), in any bipartition of G, A_0 is included in the part containing u and A_1 in the other. $A_0 \cup A_1 = V(G)$ since the graph is connected then there are only two possible partitions of G: (A_0, A_1) and (A_1, A_0). □

There are graphs which are not bipartite, for example the odd cycles. Indeed, in the cycle $(v_0, v_1, \ldots, v_{2k}, v_0)$ the path (v_0, v_{2k}) and $(v_0, v_1, \ldots, v_{2k})$ are two (v_0, v_{2k})-paths of different length parity. Hence if a graph is bipartite, it contains no odd cycles. This easy necessary condition to be bipartite is in fact sufficient.

Theorem 2.5. *A graph G is bipartite if and only if it has no odd cycle.*

Proof. Clearly, it suffices to prove it for connected graphs. Let G be a connected graph. If G contains an odd cycle, it is not bipartite.

Conversely, assume that G contains no odd cycle. Let v_0 be a vertex of G. Let A_0 (resp. A_1) be the set of vertices at even (resp. odd) distance to v_0 in G. Let us now show that (A_0, A_1) is a bipartiton of G. Let uv be an edge of G and P_u (resp. P_v) be a shortest (u, v_0)-path (resp. (v, v_0)-path). The concatenation P_u, P_v and (v, u) is a closed walk. By Proposition 1.3, this walk has even length otherwise G would contain an odd cycle. Hence P_u and P_v have different length parity and so uv has an endvertex in each of the A_i, $i = 0, 1$. □

The above proof may be translated into an algorithm which, given a connected graph G, returns either a bipartition if G is bipartite or "G is not bipartite" otherwise. Basicallly, it runs a Breadth-First Search from a vertex and check if there is no edge between vertices of levels of differents parity. Hence instead of marking the vertices with their level number as for the distance (see Subsection 2.1.1), we mark them with the parity of their level and thus we just need two marks, 0 and 1.

Algorithm 2.1 (Finding a bipartition)**.**

1. Pick a vertex x and mark it with $m(x) := 0$; $N := \{x\}$.
2. If N is non-empty, then remove a vertex v of N and do the following.

   For each neighbour w of v do
   - If $m(w) = m(v)$, return "G is not bipartite";
   - Otherwise if w is unmarked, then mark it with $m(v) + 1$ *mod* 2 and put w in N; Go to 2.
3. If N is empty, let $A_i, i = 0, 1$ be the set of vertices marked i and return "G is bipartite with bipartition" (A_0, A_1).

Algorithm 2.1 may be easily modified to return an odd cycle when G is not bipartite. See Exercise 2.10.

2.4 Exercises

Exercise 2.1. Show Theorem 2.1-(i).

Exercise 2.2 (Entriger, Kleitman and Székely)**.**
For a connected graph G, define $\sigma(G) = \sum_{u,v \in V(G)} dist(u,v)$.

1) Let G be a connected graph. For $v \in V(G)$, let T_v be a BFS-tree of G rooted at v. Show that $\sum_{v \in V(G)} \sigma(T_v) = 2(n-1)\sigma(G)$.

2) Deduce that every connected graph G has a spanning tree T such that $\sigma(T) \leq 2(1 - \frac{1}{n})\sigma(G)$.

Exercise 2.3 (Tuza)**.** Let G be a connected graph, let x be a vertex of G, and let T be a spanning tree of G that maximizes the function $\sum_{v \in V(G)} dist_T(x,v)$. Show that T is a DFS-tree of G.

Exercise 2.4 (Chartrand and Kronk)**.** Let G be a connected graph in which every DFS-tree is a path (rooted at the start). Show that G is a cycle, a complete graph, or a bipartite graph in which both parts have the same number of vertices.

Exercise 2.5 (Pósa)**.** A *chord* of a cycle C in a graph G is an edge in $E(G) \setminus E(C)$ both of whose endvertices lie on C. Let G be a graph such that $|E(G)| \geq 2|V(G)| - 3$ and $|V(G)| \geq 4$. Show that G contains a cycle with at least one chord.

Exercise 2.6. Let a be vertex of a connected graph G. Prove that G is bipartite if and only if $dist(a,b) \neq dist(a,c)$ for all edge bc.

Exercise 2.7.
1) Show that every tree is bipartite.
2) Prove that every tree has a leaf in the largest part of its bipartition.

Exercise 2.8. Prove that a bipartite graph G has at most $|V(G)|^2/4$ edges and give a graph attaining this bound.

Exercise 2.9. Show that a graph is bipartite if and only if each of its subgraphs H has a stable set of size at least $|V(H)|/2$.

Exercise 2.10. Give an algorithm that, given a connected graph G, returns either a bipartition if G is bipartite or an odd cycle if G is non-bipartite.

Exercise 2.11. Describe an algorithm based on a breadth-first search for finding a shortest odd cycle in a graph.

Exercise 2.12. Let $G = ((A,B),E)$ be a bipartite graph without isolated vertices such that $d(x) \geq d(b)$ for all $xy \in E$, where $a \in A$ and $b \in B$. Prove that $|A| \leq |B|$, with equality if and only if $d(a) = d(b)$ for all $ab \in E$.

Chapter 3

Complexity of algorithms

In this chapter, we see how problems may be classified according to their level of difficulty.

Most problems that we consider in these notes are of general character, applying to all members of some family of graphs or digraphs. By an *instance* of a problem, we mean the problem applied to one specific member of the family. For example, an instance of Algorithm 2.6 is the problem of finding all the strongly connected components of a particular digraph.

An *algorithm* for solving a problem is a well-defined procedure which accepts any instance of the problem as *input* and returns a solution to the problem as *output*. Designing computationally efficient algorithms for solving graphs problems is one of the main concern of graph theorists and computer scientists. The two aspects of theoretical interest in this regards are, firstly, to verify that a proposed algorithm does indeed perform correctly and, secondly, to analyse how efficient a procedure is. In the previous chapters, we have already encountered algorithms for solving a number of basic problems. In each case, we have established their validity and estimated their running time.

By the *computationnal complexity* (or, for short, *complexity*) of an algorithm, we mean the number of basic computational steps (such as arithmetical operations and comparisons) required for its execution. This number clearly depends on the size and nature of the input. In the case of graphs, the complexity is a function of the number of bits required to encode the adjacency list of the input graph $G = (V,E)$, a function of $|V|$ and $|E|$. (The number of bits required to encode an integer k is $\lceil \log_2 k \rceil$.) Naturally, when the input includes additional information, such as weights on the vertices or edges of the graph, this too must be taken into account in calculating the complexity. If the complexity is bounded above by a polynomial in the input size, the algorithm is called a *polynomial-time algorithm*. Such an algorithm is further qualified as *linear-time* if the polynomial is a linear function, *quadratic-time* if it is a quadratic function, and so on.

3.1 Computational complexity

Polynomial-time solvable algorithms The significance of polynomial-time algorithms is that they are usually found to be computationally feasible, even for large input graphs. By contrast,

algorithms whose complexity is exponential in the size of the input have running times which render then unusable even on inputs of moderate size.

For example, Algorithm 1.1 runs in polynomial time: The eulerian tour will be represented with a function *next* such that at the end, $next(e)$ is the edge that immediately follows e in the tour. Each time we insert an edge e, we change the value of *next* for at most two edges: the one of the edge $e_l = v_{l-1}v_l$ if $e = v_l v_{l+1}$ is not in W, or those of $v_{i-1}v_i$ and $e_l = v_{l-1}v_l$ if $e = v_i u$. But an edge is inserted exactly once so the complexity is $O(|E|)$.

The algorithms discussed in Chapter 2 also run in polynomial time. In breadth-first search, each edge is examined for possible inclusion in the tree just twice, (once per endvertex). The same is true for depth-first search. Each time that an edge is considered, a constant number of operations (test, addition or removal in a queue or a stack, mark). Hence, Algorithms 2.2 and 2.3 perform in time $O(|E|)$. The complexity of Algorithms 2.4 and 2.5 is also $O(|E|)$ since they are variation of breadth-first search, as well as the complexity of Algorithm 2.6 because it is a variation of depth-first search.

Although our analysis of these algorithms is admittedly cursory, and leaves out many pertinent details, it should be clear that they do indeed run in polynomial time. A thorough analysis of these and other graph algorithms can be found in the books by Aho et al. [2] and Papadimitriou [6]. On the other hand, there are many basic problems for which polynomial-time algorithms have yet to be found, and indeed might well not exist. Determining which problems are solvable in polynomial time and which are not is evidently a fundamental question. In this connection, a class of probems denoted by $\mathcal{NP}$ (standing for *nondeterministc polynomial-time*) plays an important role. we give here an informal definition of this class: a precise treatment can be found in the book of Garey and Johnson [4], or in Chapter 29 of the *Handbook of Combinatorics* [5].

The classes $\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$ A *decision problem* is a question whose answer is either 'yes' or 'no'. Such a problem belongs to the class $\mathcal{P}$ if there is a polynomial-time algorithm that solves any instance of the problem in polynomial time. It belongs to the class $\mathcal{NP}$ if, given any instance of the problem whose answer is 'yes', there is a certificate validating this fact which can be checked in polynomial time; such a certificate is said to be *succint*. Analogously, a decision problem belongs the the class co-$\mathcal{NP}$ if, given any instance of the problem whose answer is 'no', there is a succint certificate which confirms that this is so. It is immediate from those definitions that $\mathcal{P} \subseteq \mathcal{NP}$. Likewise, $\mathcal{P} \subseteq$ co-$\mathcal{NP}$. Thus

$$\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}.$$

Consider, for example, the problem of determining whether a graph is bipartite. This decision problem belongs to $\mathcal{NP}$, because a bipartition is a succint certificate: given a bipartition (A,B) of a bipartite graph G, it suffices to check that each edge of G has one endvertex in A and one endvertex in B. The problem also belongs to co-$\mathcal{NP}$ because, by Theorem 2.5, every nonbipartite graph contains an odd cycle, and any such cycle constitutes a succint certificate of the graph's nonbipartite character. It thus belongs to $\mathcal{NP} \cap$ co-$\mathcal{NP}$. In fact, as shown by Algorithm 2.1 which run in polynomial time, it belongs to $\mathcal{P}$.

Consider now the problem of deciding whether a graph has a *hamiltonian cycle*, that is a spanning cycle or its analog in digraph.

Problem 3.1 (Hamiltonian Cycle)**.**
Instance: A graph G.
Decide: Does G have a hamiltonian cycle?

Problem 3.2 (Directed Hamiltonian Cycle)**.**
Instance: A digraph G.
Decide: Does G have a directed hamiltonian cycle?

If the answer is 'yes', then any hamiltonian cycle would serve as a succint certificate. However, should the answer be 'no', what could consitute a succint certificate confirming this fact? In constrast to the two problems described above, no such certificate is known! In other words, notwithstanding that the Hamiltonian Cycle Problem is clearly member of the class $\mathcal{NP}$, it has not yet been shown to belong to co-$\mathcal{NP}$, and might very well not belong to this class.

We have noted three relations of inclusion among the classes $\mathcal{P}$, $\mathcal{NP}$ and co-$\mathcal{NP}$, and it is natural to ask wether these inclusiuons are proper. Because $\mathcal{P} = \mathcal{NP}$ if and only if $\mathcal{P} = \text{co-}\mathcal{NP}$, two basic questions arise, both of which have been posed as conjectures.

Conjecture 3.3.

$$\mathcal{P} \neq \mathcal{NP}$$

Conjecture 3.4 (Edmonds)**.**

$$\mathcal{P} = \mathcal{NP} \cap \text{co-}\mathcal{NP}$$

Conjecture 3.3 is one of the most fundamental open questions in all mathematics. (A prize of one million dollar has been offered for its resolution). It is widely (but not universally) believed that the conjecture is true, that there are problems in $\mathcal{NP}$ for which no polynomial-time algorithm exists. One such problem would be the Directed Hamiltonian Cycle Problem. As we show in Section 3.3, this problem is at least as hard to solve as any problem in the class $\mathcal{NP}$; more precisely, if a polynomial-time algorithm for this problem should be found, it could be adapted to solve any problem in $\mathcal{NP}$ in polynomial time by means of a suitable transformation.

Conjecture 3.4 is strongly supported by empirical evidence. Most decision problems which are known to belong to $\mathcal{NP} \cap \text{co-}\mathcal{NP}$ are also known to belong to $\mathcal{P}$. A case in point is the problem of deciding whether a given integer is prime. Although it had been known for some time that this problem belongs to both $\mathcal{NP}$ and co-$\mathcal{NP}$, a polynomial-time algorithm for testing primality was discovered only much more recently, by Agrawal, Kayal and Saxena [1].

3.2 Polynomial reductions

A common approach to problem-solving is to transform the given problem into one whose solution is already known, and then convert that solution into a solution of the original problem.

Of course, this approach is feasible only if the transformation can be made rapidly. The concept of polynomial reduction captures this requirement.

A *polynomial reduction* of a problem P to problem Q is a pair of polynomial-time algorithms which transforms each instance I of P to an instance J of Q, and the other which transforms a solution for the instance J to a solution for the instance I. If such a reduction exists, we say that P is *polynomially reducible* to Q, and write $P \preceq Q$. The significance of polynomial reducibility is that if $P \preceq Q$, and if there is a polynomial-time algorithm for solving Q, then this algorithm can be converted into a polynomial-time algorithm for solving P.

In particular, if P and Q are both decision problems, it translates into symbols:

$$P \preceq Q \text{ and } Q \in \mathcal{P} \Rightarrow P \in \mathcal{P}. \tag{3.1}$$

Observe that, since the solution of a decision problem is either 'yes' or 'no', the second algorithm which transforms a solution for the instance J to a solution for the instance I, is trivial: either it is identity (the solution to I is 'yes' if and only if the answer to J is 'yes') or the negation (the answer to I is 'yes' if and only if the answer to J is 'no').

3.3 $\mathcal{NP}$-complete problems

3.3.1 The class $\mathcal{NPC}$

We have just seen how polynomial reductions may be used to produce new polynomial-time algortihms from existing ones. By the same token, polynomial reductions may also be used to link 'hard' problems, ones for which no polynomial-time algorithm exists, as can be seen by writing (3.1) in a different form:

$$P \preceq Q \text{ and } P \notin \mathcal{P} \Rightarrow Q \notin \mathcal{P}.$$

This viewpoint led Cook and Levin to define a special class of seemingly intractable decision problems, the class of $\mathcal{NP}$-complete problems. Informally, there are the problems in the class $\mathcal{NP}$ which are 'at least as hard to solve' as any problem in $\mathcal{NP}$.

Formally, a problem P in $\mathcal{NP}$ is *NP-complete* if $P' \preceq P$ for every problem P' in $\mathcal{NP}$. The class of $\mathcal{NP}$-complete problems is denoted by $\mathcal{NPC}$. It is by no means obvious that $\mathcal{NP}$-complete problems should exist at all. On the other hand, once one such problem has been found, the $\mathcal{NP}$-completeness of other problems may be established by means of polynomial reductions, as follows.

In order to prove that a problem Q in $\mathcal{NP}$ is $\mathcal{NP}$-complete, it suffices to find a polynomial reduction to Q of some known $\mathcal{NP}$-complete problem P. Why is this so? Suppose that P is $\mathcal{NP}$-complete. Then $P' \preceq P$ for all $P' \in \mathcal{NP}$. If $P \preceq Q$, then $P' \preceq Q$ for all $P' \in \mathcal{NP}$, by the transitivity of the relation $\preceq$. In other words, Q is $\mathcal{NP}$-complete. In symbols:

$$P \preceq Q \text{ and } P \in \mathcal{NPC} \Rightarrow Q \in \mathcal{NPC}.$$

Cook and Levin made a fundamental breakthrough by showing that there do indeed exist $\mathcal{NP}$-complete problems. More precisely, they proved that the satisfiability problem for boolean

formulae is $\mathcal{NP}$-complete. We now describe this problem, and examine the theoretical and practical implications of their discovery.

3.3.2 Boolean formulae and satisfiability

A *boolean variable* is a variable which takes on one of two values, $false$ or $true$. Boolean variables may be combined into *boolean formulae*, which may be defined recursively as follows.

- Every boolean variable is a boolean formula.
- If f is a boolean formula, then so too is $(\neg f)$, the *negation* of f.
- If f_1 and f_2 are boolean formulae, then so too are:
  - $(f_1 \vee f_2)$, the *disjunction* of f_1 and f_2,
  - $(f_1 \wedge f_2)$, the *conjunction* of f_1 and f_2.

These three operations may be thought of informally as 'not f', 'f_1 or f_2', and 'f_1 and f_2', respectively.

An assignment of values to the variables is called a *truth assignment*. Given a truth assignment, the value of the formula may be computed according to the following rules:

- if $f = false$, then $(\neg f) = true$, else $(\neg f) = false$;
- if $f_1 = true$ or $f_2 = true$, then $(f_1 \vee f_2) = true$, else $(f_1 \vee f_2) = false$;
- if $f_1 = true$ and $f_2 = true$, then $(f_1 \wedge f_2) = true$, else $(f_1 \vee f_2) = false$.

Two boolean formulae are *equivalent* (written $\equiv$) if they take the same value for each assignment of the variable involved. It follows easily from the above rules that disjunction and conjunction are *commutative* and *associative*. Hence, all the formulae obtained from k subformulae $f_1, f_2, \ldots, f_k$ by means of disjunction are all equivalent. Any of these is denoted by $(f_1 \vee f_2 \vee \cdots \vee f_k)$.

A boolean formula is *satisfiable* if there is a truth assignment of its variables for which the value of the formula is $true$. Clearly, some boolean formulae are satisfiable and some are not. This poses the general problem:

Problem 3.5 (SAT ; Boolean Satisfiability)**.**
Instance: a boolean formula f.
Decide: Is f satisfiable?

Observe that SAT belongs to $\mathcal{NP}$. Indeed given an appropriate truth assignment, it can be checked in polynomial time that the value of the formula is indeed $true$.

Theorem 3.6 (Cook – Levin)**.** *The problem* SAT *is $\mathcal{NP}$-complete.*

The proof of the Cook-Levin Theorem involves the notion of a Turing machine, and is beyond the scope of these notes. A proof may be found in the books of Garey and Johnson [4] or Sipser [7].

Many combinatorial problems are shown to be $\mathcal{NP}$-complete. See for example [4] or [3]. One of the most celebrated and to which many reductions are proved, is 3-SAT. To define it, we need a few more definitions.

A variable x, or its negation $\bar{x}$, is a *literal*, and a disjunction of literals is a *clause*. Any conjuction of disjunctive clauses is referred to as a formula in *conjunctive normal form*. It can be shown that every boolean formula is equivalent, via a polynomial reduction, to one in conjunctive normal form (Exercise 3.1). Furthermore, every boolean formula is equivalent, again via a polynomial reduction, to one in conjunctive normal form with exactly three literals per clause. The decision problem for such boolean formulae is known as 3-SAT.

Problem 3.7 (3-SAT)**.**
Instance: a boolean formula f in conjunctive normal form with exactly three literals per clause.
Decide: Is f satisfiable?

Theorem 3.8. *The problem* 3-SAT *is* $\mathcal{NP}$*-complete.*

Proof. By Theorem 3.6, it suffices to prove that SAT $\preceq$ 3-SAT. Let f be a boolean formula in conjunctive normal form. We show how to construct, in polynomial time, a boolean formula f' in conjunctive normal form such that:

(i) each clause in f' has three literals;

(ii) f is satisfiable if and only if f' is satisfiable.

Such a formula f' may be obtained by the addition of new variables and clauses, as follows.

Suppose that some clause of f has just two literals, for instance the clause $(x_1 \vee x_2)$. In this case, we simply we simply replace this clause by two clauses with three literals, $(x_1 \vee x_2 \vee x)$ and $(x_1 \vee x_2 \vee \bar{x})$, where x is a new variable. Clearly $(x_1 \vee x_2)$ is equivalent to the conjunction of these two clauses.

Clauses with single literals may be dealt with in a similar manner. If x_1 is a clause, then we first replace this clause by the two clauses with two literals $(x_1 \vee x)$ and $(x_1 \vee \bar{x})$, where x is a new variable. We then replace each of these two clauses by two clauses with three literals as above.

Now suppose that some clause $(x_1 \vee x_2 \vee \cdots \vee x_k)$ of f has k literals, where $k \geq 4$. In this case, we add $k-3$ new variables $y_1, y_2, \ldots, y_{k-3}$ and form the following $k-2$ clauses, each with three literals.

$$(x_1 \vee x_2 \vee y_1), (\bar{y}_1 \vee x_3 \vee y_2), (\bar{y}_2 \vee x_4 \vee y_3), \ldots, (\bar{y}_{k-4} \vee x_{k-2} \vee y_{k-3}), (\bar{y}_{k-3} \vee x_{k-1} \vee x_k).$$

One may verify that $(x_1 \vee x_2 \vee \cdots \vee x_k)$ is equivalent to the conjunction of these $k-2$ clauses. □

3.3.3 Some $\mathcal{NP}$-completeness proofs

As we have observed, in order to show that a decision problem Q in $\mathcal{NP}$ is $\mathcal{NP}$-complete, it suffices to find a polynomial reduction to Q of a known $\mathcal{NP}$-complete problem. This is generally easier said than done. What is needed is to first decide on an appropriate $\mathcal{NP}$-complete problem P and then come up with a suitable polynomial reduction. In the case of graphs, the latter step is often achieved by means of a construction whereby certain special subgraphs, referred to as 'gadgets', are inserted into the instance of P so as to obtain an instance of Q with the required properties. We now describe an illustration of this technique by showing how 3-SAT may be reduced to the Directed Hamiltonian Cycle Problem via an intermediate problem, the Exact Cover Problem.

Let $\mathcal{A}$ be a family of subsets of a finite set X. An *exact cover* of X by $\mathcal{A}$ is a partition of X, each member of which belongs to $\mathcal{A}$. For instance, if $X = \{x_1, x_2, x_3\}$ and $\mathcal{A} = \{\{x_1\}, \{x_1, x_2\}, \{x_2, x_3\}\}$, then $(\{x_1\}, \{x_2, x_3\})$ is an exact cover of X by $\mathcal{A}$. This notion give rise to the following decision problem.

Problem 3.9 (Exact Cover)**.**
Instance: a set X and a family $\mathcal{A}$ of subsets of X.
Decide: Is there an exact cover of X by $\mathcal{A}$?

We first describe a polynomial reduction of 3-SAT to the Exact Cover Problem, and then a polynomial reduction of the Exact Cover Problem to the Directed Hamiltonian Cycle Problem. Since 3-SAT is $\mathcal{NP}$-complete by Theorem 3.8, this implies that the Exact Cover Problem and the Directed Hamiltonian Cycle Problem are $\mathcal{NP}$-complete.

Theorem 3.10. 3-SAT *is polynomially reducible to the Exact Cover Problem.*

Proof. Let f be an instance of 3-SAT, with variables $x_1, \ldots, x_n$ and clauses $f_1, \ldots, f_m$. We first construct a graph G from f by setting:

$$\begin{aligned} V(G) &= \{x_i \mid 1 \le i \le n\} \cup \{\overline{x}_i \mid 1 \le i \le n\} \cup \{f_j \mid 1 \le j \le m\}, \\ E(G) &= \{x_i \overline{x}_i \mid 1 \le i \le n\} \cup \{x_i f_j \mid x_i \in f_j\} \cup \{\overline{x}_i f_j \mid \overline{x_i} \in f_j\}, \end{aligned}$$

where the notation $x_i \in f_j$ (resp. $\overline{x_i} \in f_j$) signifies that x_i (resp. $\overline{x_i}$) is a literal of the clause f_j. We then obtain an instance $(X, \mathcal{A})$ of the Exact Cover Problem from this graph G by setting:

$$\begin{aligned} X &= \{f_j \mid 1 \le j \le m\} \cup E(G), \quad \text{and} \\ \mathcal{A} &= \{E(x_i) \mid 1 \le i \le n\} \cup \{E(\overline{x_i}) \mid 1 \le i \le n\} \cup \{\{f_j\} \cup F_j \mid F_j \subset E(f_j), 1 \le j \le m\}, \end{aligned}$$

where $E(x)$ denotes the set of edges incident to vertex x in the graph G.

It can be verified that the formula f is satisfiable if and only if the set X has an exact cover by the family $\mathcal{A}$. □

Corollary 3.11. *The Exact Cover Problem is $\mathcal{NP}$-complete.*

Theorem 3.12. *The Exact Cover Problem is polynomially reducible to the Directed Hamiltonian Cycle Problem.*

Proof. Let $(X,\mathcal{A})$ be an instance of the Exact Cover Problem, where $X = \{x_i, 1 \leq i \leq n\}$ and $\mathcal{A} = \{A_j \mid 1 \leq j \leq m\}$. We construct a digraph D as follows. Let P be a directed path whose arcs are labelled by the elements of X, Q a directed path whose arcs are labelled by the elements of $\mathcal{A}$, and for $1 \leq j \leq m$, R_j a directed path whose vertices are labelled by the elements of A_j. The paths P, Q, and R_j, $1 \leq j \leq m$, are assumed to be pairwise disjoint. We add an arc from the start of P to the start of Q, and from the terminus of Q to the terminus of P. For $1 \leq j \leq m$, we also add an arc from the tail of the arc labelled A_j of Q to the start of R_j, and from the terminus of R_j to the head of the arc labelled A_j of Q.

For $1 \leq j \leq m$, we now transform the directed path R_j into a digraph D_j by replacing each vertex labelled x_i of R_j by a *symmetric path* P_{ij} of length two, that is the symmetric digraph obtained from a path of length two by replacing each edge by two arcs in each direction. Moreover, for every such symmetric path, we add an arc from the start of $P_{i,j}$ to the tail of the arc labelled x_i of P, and one from the head of x_i to the terminus of $P_{i,j}$. We denote the resulting digraph by D. See Figure 3.1.

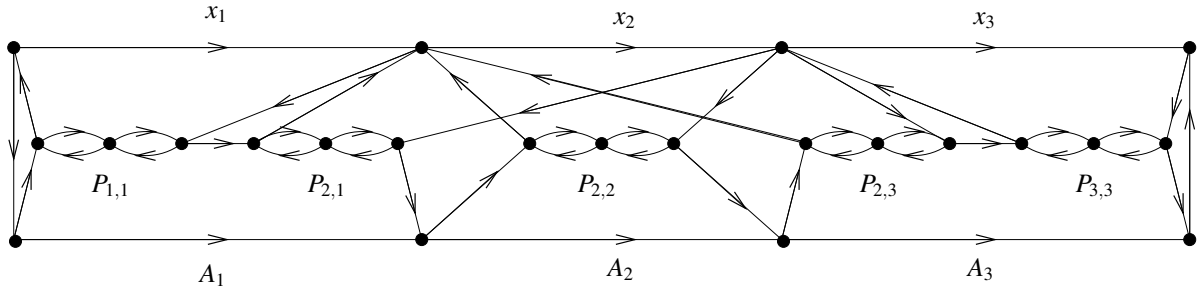


Figure 3.1: The digraph D when $X = \{x_1, x_2, x_3\}$ and $\mathcal{A} = \{\{x_1, x_2\}, \{x_2\}, \{x_2, x_3\}\}$.

Observe now, that the digraph D has a directed hamiltonian cycle C if and only if the set X has an exact cover by $\mathcal{A}$.

Suppose first that D has a directed hamiltonian cycle C. If C does not use the arc labelled A_j in Q, it is obliged to traverse D_j from its start to its terminus. Conversely, if C uses the arc labelled A_j in Q, it is obliged to include each one of the paths $P_{i,j}$ in D_j in its route from the start of P to the terminus of P. Moreover, C traces exactly one of the paths $P_{i,j}$ (for $x_i \in A_j$) in travelling from the head of the arc of P labelled x_i to its tail. Hence A_j that label the arcs of $Q \cap C$ form a partition of X.

Reciprocally, if we have an exact cover of X by $\mathcal{A}$, a directed hamiltonian cycle C of D may be constructed by taking the arcs labelled A_j when A_j is not a member of the exact cover and all the paths P_{ij} for X_j in the cover and $x_i \in X_j$ and completing by adding arcs in an obvious way.

Finally, the number of vertices of D is

$$|E(D)| = |X| + |\mathcal{A}| + 3\sum_{j=1}^{m} |A_J| + 2.$$

This function is bounded above by a linear function of the size of the instance $(X,\mathcal{A})$, so the above reduction is indeed polynomial. □

Corollary 3.13. *The Directed Hamiltonian Cycle is $\mathcal{NP}$-complete.*

A huge number of decision problems have been shown to be $\mathcal{NP}$-complete. See for example the book of Garey and Johnson [4].

3.4 $\mathcal{NP}$-hard problems

We now turn to the computational complexity of optimization problems such as the Travelling Salesman Problem. An *edge-weighted graph* is a pair (G,w) where $G=(V,E)$ is a graph and $w : E \to \mathbb{R}$ is a *weight function*.

Problem 3.14 (Travelling Salesman)**.**
Instance: an edge-weighted complete graph (G,w).
Find: a hamiltonian cycle C of G of minimum weight, i.e. such that $\sum_{e\in E(C)} w(e)$ is minimum.

This problem contains the Hamiltonian Cycle Problem as a special case. To see this, associate with a given graph G the edge-weighted complete graph on $V(G)$ in which the weight function w is defined by $w(uv)=0$ if $uv \in E(G)$, and $w(uv)=1$ otherwise. The resulting edge-weighted complete graph has a hamiltonian cycle of weight zero if and only if G has a hamiltonian cycle. Thus, any algorithm for solving the Travelling Salesman Problem will also solve the Hamiltonian Cycle Problem, and we may conclude that the former problem is at least as hard as the latter. Because the Hamiltonian Cycle Problem is $\mathcal{NP}$-complete, (See Exercise 3.2), the Travelling Salesman Problem is at least as hard as any problem in $\mathcal{NP}$. Such problems are called *$\mathcal{NP}$-hard*.

Observe that every optimization problem implicitely includes an infinitude of decision problems. For example, the Travelling Salesman Problem includes, for each real number r, the following decision problem. Given an edge-weighted graph (G,w), is there a hamiltonian cycle of weight at most r? If one of these problems is $\mathcal{NP}$-complete, then the optimization problem is $\mathcal{NP}$-hard. However, the problem may still be $\mathcal{NP}$-hard even if all these problems are polynomial-time solvable. For example, it is the case for the following basic problem.

Problem 3.15 (Maximum Clique)**.**
Instance: a graph G.
Find: a clique of maximum size in G.

If k is a fixed integer not depending on $|V(G)|$, the existence of a k-clique can be decided in polynomial time, simply by means of an exhaustive search, because the number of k-subsets of $V(G)$ is bounded above by $|V(G)|^k$. However, if k depends on $|V(G)|$, this is no longer true. Indeed, the problem of deciding whether a graph G has a k-clique, where k depends on $|V(G)|$ is $\mathcal{NP}$-complete.

Theorem 3.16. *The following problem is $\mathcal{NP}$-complete.*
Instance: a graph G.
Decide: $\omega(G) \geq |V(G)|/7$?

Corollary 3.17. *The Maximum Clique Problem is $\mathcal{NP}$-hard.*

Since a subset S of $V(G)$ is a stable set in G if and only if S is a clique in $\overline{G}$, the following problem is polynomially equivalent to the Maximum Clique Problem, and thus is $\mathcal{NP}$-hard also.

Problem 3.18 (Maximum Stable Set)**.**
Instance: a graph G.
Find: a stable set of maximum size in G.

A huge collection of optimization problems have been shown to be $\mathcal{NP}$-hard, see [**?**].

3.5 Approximation algorithms

For $\mathcal{NP}$-hard optimization problems of practical interest, such as the Travelling Salesman Problem, the best that one can reasonably expect of a polynomial-time algorithm is that it should always return a feasible solution which is not too far from optimality.

Given a real number $r \geq 1$, an *r-approximation algorithm* for a minimization problem is an algorithm that returns a feasible solution whose value is no more than r times the optimal value; similarly, an *r-approximation algorithm* for a maximization problem is an algorithm that returns a feasible solution whose value is no less than r times the optimal value; the smaller the value of r, the better the approximation. Naturally, the running time of the algorithm is an equally important factor. We give an example.

Problem 3.19 (Maximum Cut)**.**
Instance: a graph G.
Find: a spanning bipartite subgraph F of G with the maximum number of edges.

It can be shown that the Maximum Cut Problem is $\mathcal{NP}$-hard (Exercise 3.3).

Theorem 3.20. *The Maximum Cut Problem admits a polynomial-time* 2*-approximation algorithm.*

Proof. We now describe an algorithm that find a bipartite subgraph F such that $|E(F)| \geq |E(G)|/2$. Since an subgraph of G cannot have more than $|E(G)|$ edges, this is a 2-approximation algorithm.

Algorithm 3.1 (Maximum-Cut Approximation)**.**

2. Take any ordering $v_1, v_2, \ldots, v_n$ of the vertices; $A := \emptyset$; $B := \emptyset$; $E := \emptyset$.

2. For $i = 1$ to n, if v_i has more neighbours in A than in B, then add v_i to B and all edges joining v_i to elements of A to E. Else add v_i to A and all edges joining v_i to elements of B to E.

3. Return $((A,B),E)$.

This algorithm examined every vertex exactly once and each time its examines a vertex it must counts the numbers of vertices in A and B and compare them. Then its complexity is at most $O(|V|^2)$.

Let us now show that the returned bipartite graph F satisfies $|E(F)| \geq |E(G)|/2$. Therefore let us denote by $F_i = ((A_i, B_i), E_i)$ the bipartite graph constructed after step i that is after having examing v_i at Step 2 in the above algorithm. and $G_i = G\langle\{v_1, \ldots, v_i\}\rangle$. We prove by induction that $|E(F_i)| \geq |E(G_i)|/2$, the result holding vacuously when $i = 0$. Suppose now that $i > 0$ and that the result holds for $i-1$. We obtained F_i from F_{i-1} by adding V_i to the part in which it has the smaller number of neighbours. Hence, the number of edges incident to v_i that we add to F_{i-1} is at least has large has half the number of edges joining v_i to a vertex in $V(F_{i-1}) = \{v_1, \ldots, v_{i-1}\}$. Since $|E(F_{i-1})| \geq |E(G_{i-1})|/2$ by the induction hypothesis, we have $|E(F_i)| \geq |E(G_i)|/2$. □

The analog of the Maximum Cut Problem in edge-weighted graph, called the Weighted Maximum Cut Problem also admits a polynomial-time 2-approximation algorithm. See Exercise 3.4.

If some algorithms admits polynomial-time approximation algorithms, some others do not. For example, this is the case for the Travelling Salesman Problem: for any $t \geq 2$, there cannot exists a polynomial-time t-approximation algorithm for solving the Travelling Salesman Problem, unless $\mathcal{P} = \mathcal{NP}$. (Exercise 3.5). However some special cases the Travelling Salesman Problem admit polynomial-time approximation algorithm. For example, such an algorithm, when the weights satisfies the triangle inequality, is discussed in Section 4.2.3.

For more on approximation algorithms, we refer the interested reader to the book of Vazirani [8].

3.6 Exercises

Exercise 3.1. Let f_1 and f_2 be two boolean formulae in conjuctive normal form.
1) Show that:

a) $f_1 \wedge f_2$ is in conjunctive normal form;

b) $f_1 \vee f_2$ is equivalent to a boolean formula in conjunctive normal form;

c) $\neg f_1$ is equivalent to a boolean formula in conjunctive normal form;

2) Deduce that every boolean formula is equivalent to a boolean formula in conjunctive normal form.

Exercise 3.2.
1) Describe a polynomial-time reduction of the Directed Hamiltonian Cycle Problem to the Hamiltonian Cycle Problem.
2) Deduce the Hamiltonian Cycle Problem is $\mathcal{NP}$-complete.

Exercise 3.3. Show that the Maximum Cut Problem is $\mathcal{NP}$-hard.

Exercise 3.4. Describe a polynomial-time 2-approximation algorithm for the following problem, called the Weighted Maximum Cut Problem.
Instance: an edge-weighted graph (G,w).
Find: a spanning bipartite subgraph F of G with the maximum weight.

Exercise 3.5.
1) Let G be a graph on n vertices, $n \geq 3$ vertices, and let t be a positive integer. Consider the edge-weighted complete graph (K,w), where $V(K) = V(G)$, in which $w(e) = 1$ if $e \in E(G)$ and $w(e) = (t-1)n+2$ if $e \in E(K) \setminus E(G)$. Show that:

a) (K,w) has a hamiltonian cycle of weight n if and only if G has a hamiltonian cycle;

b) any hamiltonian cycle of (K,w) of weight greater than n has weight at least $tn+1$.

2) Deduce that, unless $\mathcal{P} \neq \mathcal{NP}$, there cannot exist a polynomial-time t-approximation algorithm for solving the Travelling Salesman Problem.

Bibliography

[1] M. Agrawal, N. Kayal and N. Saxena. PRIMES is in P. *Ann. of Math. (2)* 160:781–793, 2004.

[2] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, Reading, MA, 1975. Second printing.

[3] P. Crescenzi and V. Kann, eds. *A $\mathcal{NP}$-compendium of $\mathcal{NP}$ optimization problems*. `http://www.csc.kth.se/ viggo/problemlist/`.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[5] R. L. Graham, M. Grötschel and L. Lovász, eds. *Handbook of Combinatorics*. Vol. 1,2. Elsevier, Amsterdam, 1995.

[6] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

[7] M. Sipser. *Introduction to the Theory of Computation.* Second edition. Course Technology, Boston, MA, 2005.

[8] V. V. Vazirani. *Approximation Algorithms*. Springer, Berlin, 2001.

Chapter 4

Algorithms in edge-weighted graphs

Recall that an *edge-weighted graph* is a pair (G,w) where $G=(V,E)$ is a graph and $w : E \to \mathbb{R}$ is a *weight function*. Edge-weighted graphs appear as a model for numerous problems where places (cities, computers,...) are linked with links of different weights (distance, cost, throughput,...). Note that a graph can be viewed as an edge-weighted graph where all edges have weight 1.

Let (G,w) be an edge-weighted graph. For any subgraph H of G, the *weight* of H, denoted by $w(H)$, is the sum of all the weights of the edges of H. In particular, if P is a path, $w(P)$ is called the *length* of P. The *distance* between two vertices u and v, denoted by $dist_{G,w}(u,v)$, is the length of a shortest (with minimum length) (u,v)-path.

Observe that $dist_{G,w}$ is a distance: it is *symmetric*, that is, $dist_{G,w}(u,v) = dist_{G,w}(v,u)$, and it satisfies the *triangle inequality*: for any three vertices x, y and z, $dist_{G,w}(x,z) \leq dist_{G,w}(x,y) + dist_{G,w}(y,z)$.

4.1 Computing shortest paths

Given an edge-weighted graph (G,w), one of the main problems is the computation of $dist_G(u,v)$ and finding a shortest (u,v)-path. We have seen in Subsection 2.1.1, that if all the edges have same weight then one can compute a shortest (u,v)-path by running a breadth-first search from u. Unfortunately, this approach fails for general edge-weighted graphs. See Exercise 4.1. We now describe algorithms to solve this problem in general. For this purpose, we solve the following more general problem.

Problem 4.1 (Shortest-paths tree)**.**
Instance: an edge-weighted graph (G,w) and a vertex r.
Find: a subtree T of G such that $\forall x \in V(G), dist_{G,w}(r,x) = dist_{T,w}(r,x)$.

Such a tree is called a *shortest-paths tree*.

4.1.1 Dijkstra's Algorithm

Dijkstra's Algorithm is based on the following principle. Let $S \subset V(G)$ containing r and let $\bar{S} = V(G) \setminus S$. If $P = (r, s_1, \ldots, s_k, \bar{s})$ is a shortest path from r to $\bar{S}$, then $s_k \in S$ and P is a shortest path from r to $\bar{s}$. Hence,

$$dist(r, \bar{s}) = dist(r, s_k) + w(s_k \bar{s})$$

and the distance from r to $\bar{S}$ is given by the following formula

$$dist(r, \bar{S}) = \min_{u \in S, v \in \bar{S}} \{dist(r, u) + w(uv)\}$$

To avoid to many calculations during the algorithm, each vertex $v \in V(G)$ is associated to a function $d'(v)$ which is an upper bound on $dist(r, v)$, and to a vertex $p(v)$ which is the potential parent of v in the tree. At each step, we have:

$$\begin{aligned} d'(v) &= dist(r, v) \text{ if } v \in V(T_i) \\ d'(v) &= \min_{u \in V(T_{i-1})} \{dist(r, u) + w(uv)\} \text{ if } v \in \overline{V(T_i)} \end{aligned}$$

Algorithm 4.1 (Dijkstra)**.**

1. Initialize $d'(r) := 0$ and $d'(v) := +\infty$ if $v \neq r$. T_0 is the tree consisting of the single vertex r, $u_0 := r$ and $i := 0$.
2. For any $v \in \overline{V(T_i)}$, if $d'(u_i) + w(u_i v) \leq d'(v)$, then $d'(v) := d'(u_i) + w(u_i v)$ and $p(v) := u_i$.
3. Compute $\min\{d'(v) \mid v \in \overline{V(T_i)}\}$. Let u_{i+1} a vertex for which this minimum is reached. Let T_{i+1} be the tree obtained by adding the vertex u_{i+1} and the edge $p(u_{i+1})u_{i+1}$.
4. If $i = |V| - 1$, return T_i, else $i := i + 1$ and go to Step 2.

Remark 4.2. The algorithm does not work if some weights are negative.

Complexity of Dijkstra's Algorithm: To every vertex is associated a temporary label corresponding to $(d'(v), p(v))$. They are depicted in Figure 4.1. We do

- at most $|E|$ updates of the labels;
- $|V|$ searches for the vertex v for which $d'(v)$ minimum and as many removal of labels.

The complexity depends on the choice of the data structure for storing the labels: if it is a list, the complexity is $O(|E||V| + |V|^2)$. But it can be improved using better data structures. For example, a data structure known as *heap* is commonly used for sorting elements and their

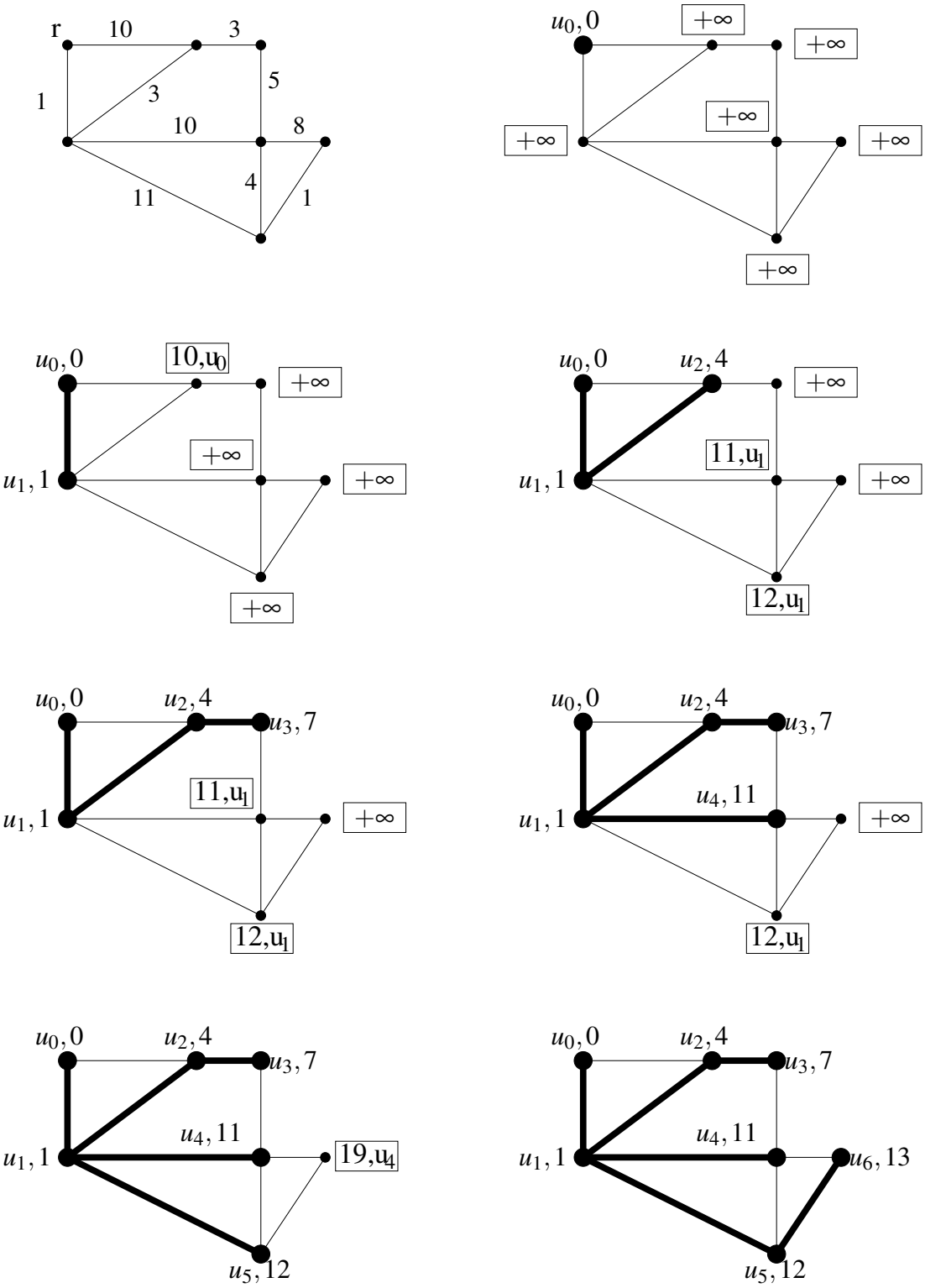


Figure 4.1: A run of Dijkstra's Algorithm on the edge-weighted graph depicted top left. At each step, bold vertices and edges are those of T_i. To each vertex t of T_i is associated its name and the value $d'(t) = dist(r,t)$. Next to each vertex v not in $V(T_i)$ is a box containing the value $d'(v)$ and $p(v)$ if $d'(v) \neq +\infty$.

associated values, called *keys* (such as edges and their weights). A heap is a rooted binary tree T **should we define it?** whose vertices are in one-to-one correspondence with the elements in question (in our case, vertices or edges). The defining property of a heap is that the key of the element located at vertex v of T is required to be at least as small as the keys of the elements located at vertices of the subtree of T rooted at v. This condition implies, in particular, that the key of the element at the root of T is one of smallest value; this element can thus be accessed instantly. Moreover, heaps can be reconstituted rapidly following small modifications such as the addition of an element, the removal of an element, or a change in the value of a key. A *priority queue* is simply a heap equipped with procedures for performing such readjustments rapidly.

Using a priority queue, the complexity of Dijkstra's Algorithm is $O(|E|\log|V|+|V|\log|V|)$.

It should be evident that data structures play a vital role in the efficiency of algorithms. For further information on this topic, we refer the reader to [4, 1, 5, 3].

4.1.2 Bellmann-Ford Algorithm

The algorithm performs n iterations, and gives a label $h(v)$ to any vertex. At iteration i, $h(v)$ is the minimum weight of a path using at most i edges between r and v.

Note that, it always exists a shortest walk using at most $|V(G)|-1$ edges between r and v (otherwise the walk would contain a cycle of negative weight).

Algorithm 4.2 (Bellmann-Ford)**.**

1. Initialization : $h(r) := 0$, $h(v) := +\infty, \forall v \neq r$.
2. For $i = 0$ to $|V(G)|-1$ do :
   for all $v \in V(G), h(v) := \min(h(v), \min\{h(u)+w(uv) \mid uv \in E(G)\})$.
3. Return $d(r,v) = h(r,v)$.

Complexity of Bellmann-Ford's Algorithm: Each iteration costs $O(|E|)$ (all edges are considered), so the total complexity is $O(|E||V|)$.

The algorithm works even if some edges have negative weight. It can also detect cycles with negative weight. There is such a cycle if and only if, after the $|V|^{\text{th}}$ iteration, the labels h may decrease. Finally, if during an iteration, no $h(v)$ decreases, then $h(v) = d(r,v)$. It is possible to improve the algorithm by continuing the iteration only if $h(v)$ becomes $\min\{h(u)+w(uv) \mid uv \in E(G)\}$ for at least one vertex. The algorithm run in time $O(L|E|)$ where L is the maximum number of edges in a shortest path.

4.2 Minimum-weight spanning tree

Another important problem is the following: given a connected edge-weighted graph, what is the connected spanning subgraph with minimum weight? If all weights are non-negative, since any connected graph has a spanning tree (Corollary 1.10), the problem consists of finding a spanning tree with minimum weight.

Problem 4.3 (Minimum-Weight Spanning Tree)**.**
Instance: a connected edge-weighted graph (G,w).
Find: a spanning tree T of G with minimum weight, i.e. for which $\sum_{e\in T} w(e)$ is minimum.

For $S \subset V(G)$, an edge $e = xy$ is *S-transversal*, if $x \in S$ and $y \notin S$. The algorithms to find a minimum-weight spanning tree are based on the fact that a transversal edge with minimum weight is contained in a minimum-weight spanning tree.

Lemma 4.4. *Let (G,w) be an edge-weighted graph and let $S \subset V$. If $e = s\bar{s}$ is an S-transversal edge with minimum weight, then there is a minimum-weight spanning tree containing e.*

Proof. Let T be a tree that does not contains e. There is a path P between s and $\bar{s}$ in T. At least one edge of P, say e', is S-transversal. Hence, the tree $T' = (T \setminus e') \cup \{e\}$ has weight $w(T') = w(T) + w(e) - w(e') \leq w(T)$ since $w(e) \leq w(e')$. Therefore, if T is a minimum spanning tree, then so does T' and $w(e) = w(e')$. □

In particular, Lemma 4.4 implies that if e is an edge of minimum weight, i.e., $w(e) = \min_{f\in E(G)} w(f) = w_{min}$, then there is a minimum-weight spanning tree containing e.

4.2.1 Jarník-Prim Algorithm

The idea is to grow up the tree T with minimum weight by adding, at each step, a $V(T)$-transversal edge with minimum weight. At each step, E_T is the set of the $V(T)$-transversal edges.

Algorithm 4.3 (Jarník-Prim)**.**

1. Initialize the tree T to any vertex x and E_T is the set of edges incident to x.
2. While $V(T) \neq V(G)$:
   Find an edge $e \in E_T$ with minimum weight. Add e and its end not in T to T. Let E_y be the set of edges incident to y. Remplace E_T by $(E_T \triangle E_y)$.

Complexity of Jarník-Prim Algorithm: During the execution, at most $|E(G)|$ edges are added in E_T, and at most $|E(G)|$ edges are removed. Indeed, an edge e is removed when both its endvertices are in $V(T)$. Since $V(T)$ grows up, e will not be added anymore to E_T. $|V(G)|$ selections of the edge of E_T with minimum weight must be performed. To performs such an algorithm we need a data structure allows the insertion, the removal and the selection of the minimum-weight element efficiently. Using a priority queue, the total complexity of Jarník-Prim Algorithm is $O(|E|\log|E|)$.

4.2.2 Boruvka-Kruskal Algorithm

Boruvka-Kruskal Algorithm is close to Jarník-Prim Algorithm and its correctness also comes from Lemma 4.4. The idea is to start from a spanning forest and to make its number of connected components decreases until a tree is obtained. Initially, the forest has no edges and, at each step, an edge with minimum weight that links two components is added.

For this purpose, we need a fast mechanism allowing to test whether or not u and v are in the same component. A way to do so consists in associating to each connected component the list of all the vertices it contains. To every vertex u is associated a vertex $p(u)$ in the same component. This vertex $p(u)$ is a *representative* of this component. It points to the set $C_{p(u)}$ of vertices of this component and to the size $size(p(u))$ corresponding to the size it.

Algorithm 4.4 (Kruskal)**.**

1. Initalize $T : V(T) := V(G)$, $E(T) := \emptyset$. Order the edges in increasing order of the weights and place them in a stack L; For all $u \in V(G)$, do $p(u) := C_u$ and $size(C_u) := 1$.
2. If $L = \emptyset$, terminate. Else, pull the edge $e = uv$ with minimum weight;
3. If $p(u) = p(v)$ (the vertices are in the same component), then go to 2. Else $p(u) \neq p(v)$, add e in T.
4. If $size(p(u)) \geq size(p(v))$, then $C_{p(u)} := C_{p(u)} \cup C_{p(v)}$, $size(p(u)) := size(p(u)) + size(p(v))$, and for any $w \in C_{p(v)}$, $p(w) := p(u)$.
   Else $(size(p(u)) < size(p(v)))$, $C_{p(v)} := C_{p(v)} \cup C_{p(u)}$, $size(p(v)) := size(p(u)) + size(p(v))$, and for any $w \in C_{p(u)}$, $p(w) := p(v)$.
5. Go to 2.

Complexity of Boruvka-Kruskal Algorithm Ordering the edges takes time $O(|E(G)|\log|E(G)|)$. Then, each edge is considered only once and deciding whether the edge must be added to the tree or not takes a constant number of operations.

Now, let us consider the operations used to update the data structure when an edge is inserted in the tree.

We do the union of to sets $C_{p(u)}$ and $C_{p(v)}$. If this sets are represented as lists with a pointor to its last element, it takes a constant time. Such unions are done $|V(G)|-1$ times.

We also have to update the values of some $p(w)$. Let $x \in V(G)$ and let us estimate the number of updates of $p(x)$ during the execution of the algorithm. Observe that when $p(x)$ is updated, the component of x becomes at least twice bigger. Since, at the end, x belongs to a component of size $|V(G)|$, then $p(x)$ is updated at most $\log_2(|V(G)|)$ times. In total, there are at most $|V(G)|\log_2|V(G)|$ such updates.

Since $|V(G)| \leq |E(G)|+1$, the total time complexity is $O(|E(G)|\log|E(G)|)$.

4.2.3 Application to the Travelling Salesman Problem

Rosenkrantz, Sterns and Lewis considered the special case of the Travelling Salesman Problem (3.14) in which the weights satisfy the *triangle inequality*: $w(xy)+w(yz) \geq w(xz)$, for any three vertices x, y and z.

Problem 4.5 (Metric Travelling Salesman)**.**
Instance: an edge-weighted complete graph (G,w) whose weights satisfy the triangle inequality.
Find: a hamiltonian cycle C of G of minimum weight, i.e. such that $\sum_{e\in E(C)} w(e)$ is minimum.

This problem is $\mathcal{NP}$-hard (see Exercise 4.11) but a polynomial-time 2-approximation algorithm using minimum-weight spanning tree exists.

Theorem 4.6 (Rosenkrantz, Sterns and Lewis)**.** *The Metric Travelling Salesman Problem admits a polynomial-time* 2*-approximation algorithm.*

Proof. Applying Jarník-Prim or Boruvka-Kruskal algorithm, we first find a minimum-weight spanning tree T of G. Suppose that C is a minimum-weight hamiltonian cycle. By deleting any edge of C we obtain a hamiltonian path P of G. Because P is a spanning tree, $w(T) \leq w(P) \leq w(C)$.

We now duplicate each edge of T, thereby obtaining a connected eulerian multigraph H with $V(H)=V(G)$ and $w(H)=2w(T)$. The idea is to transform H into a hamiltonian cycle of G, and to do so without increasing its weight. More precisely, we construct a sequence $H_0, H_1, \ldots, H_{n-2}$ of connected eulerian multigraphs, each with vertex set $V(G)$, such that $H_0 = H$, H_{n-2} is a hamiltonian cycle of G, and $w(H_{i+1}) \leq w(H_i)$, $0 \leq i \leq n-3$. We do so by reducing the number of edges, one at a time, as follows.

Let C_i be an eulerian tour of H_i, where $i < n-2$. The multigraph H_i has $2(n-2)-i > n$ edges, and thus a vertex v has degree at least 4. Let xe_1ve_2y be a segment of the tour C_i; it will follow by induction that $x \neq y$. We replace the edges e_1 and e_2 of C_i by a new edge e of weight $w(xy)$ linking x and y, thereby bypassing v and modifying C_i to an eulerian tour C_{i+1} of $H_{i+1} = (H_i \setminus \{e_1,e_2\}) \cup \{e\}$. By the triangle inequality, we have $w(H_{i+1}) = w(H_i) - w(e_1) - w(e_2) + w(e) \leq w(H_i)$. The final graph H_{n-2}, being a connected eulerian graph on n vertices and n edges, is a hamiltonian cycle of G. Furthermore, $w(H_{n-2}) \leq w(H_0) = 2w(T) \leq 2w(C)$. □

A $\frac{3}{2}$-approximation algorithm for the Metric Travelling Salesman Problem was found by Christofides [2].

4.3 Algorithms in edge-weighted digraphs

Computing shortest paths in directed graphs can be done in much the same way as in undirected graphs by growing arborescences rather than trees. Dijkstra's Algorithm and Bellman-Ford Algorithm translates naturally.

The Minimum-Weight Spanning Tree Problem is equivalent to finding the minimum-weight spanning connected subgraph. The corresponding problem in digraph, namely, finding a connected subdigraph with minimum weight in a connected digraph is much more complex. Actually, this problem is $\mathcal{NP}$-hard even when all edges have same weight because it contains the Directed Hamiltonian Cycle Problem as special case. One can easily describe a polynomial-time 2-approximation algorithm. (See Exercise 4.12). Vetta [6] found a polynomial-time $\frac{3}{2}$-approximation algorithm.

4.4 Exercices

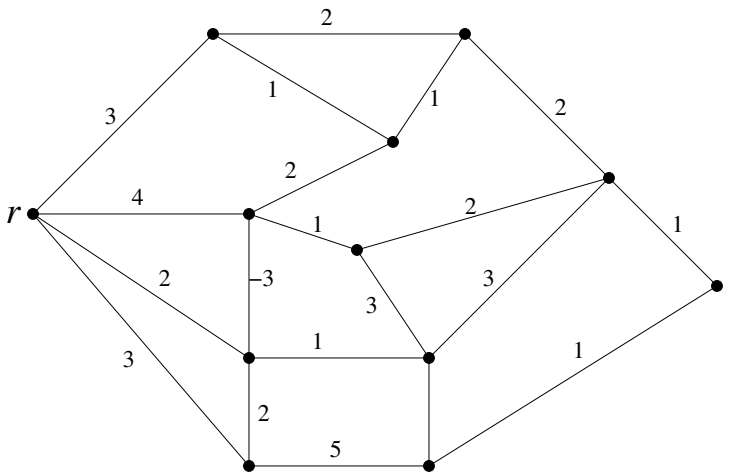
Exercise 4.1. Show a edge-weighted graph G having a vertex u such that no breadth first seach tree from u is a shortest-paths tree.

Exercise 4.2.

Consider the graph depicted in Figure 4.2.

1) Apply Dijkstra's and Bellmann-Ford algorithms for finding a shortest-paths tree from r.

2) Apply the algorithms for finding a minimum-weight spanning tree.



Exercise 4.3. Let (G,w) be a connected edge-weighted graph.
1) Prove that if w is a constant function then every shortest-paths tree is a minimum-weight

spanning tree.
2) Exhibit a connected edge-weighted graph in which there is a shortest-paths tree which is not a minimum-weight spanning tree.

Exercise 4.4. Four imprudent walkers are caught in the storm and nights. To reach the hut, they have to cross a canyon over a fragile rope bridge which can resist the weight of at most two persons. In addition, crossing the bridge requires to carry a torch to avoid to step into a hole. Unfortunately, the walkers have a unique torch and the canyon is too large to throw the torch across it. Due to dizziness and tiredness, the four walkers can cross the bridge in 1, 2, 5 and 10 minutes. When two walkers cross the bridge, they both need the torch and thus cross the bridge at the slowest of the two speeds.

With the help of a graph, find the minimum time for the walkers to cross the bridge.

Exercise 4.5. Let T be a minimum-weight spanning tree of an edge-weighted graph (G,w) and T' another spanning tree of G (not necessarily of minimum weight). Show that T' can be transformed into T by successively exchanging an edge of T' by an edge of T so that at each step the obtained graph is a tree and so that the weight of the tree never increases.

Exercise 4.6. Little Attila proposed the following algorithm to solve the Minimum-Weight Spanning Tree Problem: he considers the edges successively in decreasing order with respect to their weight and suppress the ones that are in a cycle of the remaining graph. Does this algorithm give an optimal solution to the problem? Justify your answer.

Exercise 4.7. Let (G,w) be an edge-weighted graph. For all $t \geq 1$, a *t-spanner* of (G,w) is a spanning edge-weighted graph (H,w) of (G,w) such that, for any two vertices u,v, $dist_{H,w}(u,v) \leq t \times dist_{G,w}(u,v)$.

1) Show that (G,w) is the unique 1-spanner of (G,w).
2) Let $k \geq 1$. Prove that the following algorithm returns a $(2k-1)$-spanner of (G,w).

1. Initalise H : $V(H) := V(G)$, $E(H) := \emptyset$. Place the edges in a stack in increasing order with respect to their weight. The minimum weight edge will be on top of the stack.

2. If L is empty then return H. Else remove the edge uv from the top of the stack;

3. If in H there is no (u,v)-path with at most $2k-1$ edges, add e to H.

4. Go to 2.

3) Show that the spanner returned by the above algorithm contains a minimum-weight spanning tree. (*One could show that at each step the connected components of H and the forest computed by Boruvka-Kruskal Algorithm are the same.*)

Exercise 4.8.
We would like to determine a spanning tree with weight close to the minimum. Therefore we study the following question: What is the complexity of the Minimum-Weight Spanning Tree Problem when all the edge-weights belong to a fixed set of size s. (One could consider first the case when the edges have the same weight or weight in $\{1,2\}$.

We assume that the edges have integral weights in $[1,M]$. We replace an edge with weight in $[2^i, 2^{i+1}-1]$ by an edge of weight 2^i. (We *sample* the weight.) Prove that if we compute a minimum-weight spanning tree with the simplified weight then we obtain a tree with weight at most twice the minimum for the original weight.

What happens if we increase the number of sample weights?

Exercise 4.9. 1) Let G be 2-connected edge-weighted graph. (See Chapter 5 for the definition of 2-connectivity.) Show that all the spanning trees have minimum weight if and only if all the edges have the same weight.

2) Give an example of a connected edge-weighted graph for which all the spanning tree have the same weight but whose edges do not all have the same weight.

Exercise 4.10. The *diameter* of an edge-weighted graph (G,w) is the maximum distance between two vertices: $diam(G) := \max\{dist_{G,w}(u,v) \mid u \in V(G), v \in V(G)\}$.
Show that the following algorithm computes the diameter of an edge-weighted tree T.

1. Pick a vertex x of T.
2. Find a vertex y whose distance to x is maximum (*using Dijkstra's Algorithm for example*).
3. Find a vertex z whose distance to y is maximum.
4. Return $dist_{T,w}(y,z)$.

Exercise 4.11. Show that the Metric Travelling Salesman Problem is $\mathcal{NP}$-hard.

Exercise 4.12.

1) Let D be a strongly connected digraph on n vertices. A spanning subdigraph of D is *strong-minimal* if it is strongly connected and every spanning proper subdigraph is not strongly connected.

a) Show that in the handle decomposition of a strong-minimal spanning subdigraph all the handles have length at least 2.

b) Deduce that a strong-minimal spanning subdigraph of D has at most $2n-2$ arcs.

2) Describe a polynomial-time 2-approximation for the following problem:
Instance: a strongly connected digraph D.
Find: a strongly connected spanning subdigraph with minimum number of arcs.

Exercise 4.13. An *arborescence* is an orientation of a tree in which every vertex has indegree 1 except one called the root which has indegree 0. The aim of this exercise is to obtain a polynomial-time algorithm for finding a minimum-weight spanning arborescence with prescribed root u in an edge-weighted strong digraph (D,w).

1) Show that if xy is an arc with minimum weight in (D,w) and $y \neq u$, then there is a minimum-weight spanning arborescence with root u containing xy.

2) Deduce a polynomial-time algorithm for finding a minimum-weight spanning arborescence with root u in (D,w).

Bibliography

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, Reading, MA, 1983.

[2] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. *Management Sciences Research Report* 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein. *Introduction to Algorithms.* Second Edition. MIT Press, Cambridge, MA, 2001.

[4] D. E. Knuth. *The Art of Computer Programming. Vol. 1: Fundamental Algorithms.* Addison-Wesley, Reading, MA, 1969. Second printing.

[5] R. E. Tarjan. *Data Structures and Networks Algorithms*. CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 44, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1983.

[6] A. Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. *Proceedings of the twelfth annual ACM-SIAM symposium on discrete algorithms (SODA 2001)*, pages 417–426, 2001.

Chapter 5

Connectivity

5.1 Introduction

The (strong) connectivity corresponds to the fact that a (directed) (u,v)-path exists for any pair of vertices u and v. However, imagine that the graphs models a network, for example the vertices correspond to computers and edges to links between them. An important issue is that the connectivity remains even if some computers and links fail. This will be measured by the notion of k-connectivity.

Let G be a graph. Let W be an set of edges (resp. of vertices). If $G \setminus W$ (resp. $G - W$) is not connected, then W *separates* G, and W is called an *edge-separator* (resp. *vertex-separator*) or simply *separator* of G.

For any $k \geq 1$, G is *k-connected* if it has order at least $k+1$ and no set of $k-1$ vertices is a separator. In particular, the complete graph K_{k+1} is the only k-connected graph with $k+1$ vertices. The *connectivity* of G, denoted by $\kappa(G)$, is the maximum integer k such that G is k-connected. Similarly, a graph is *k-edge connected* if it has at least two vertices and no set of $k-1$ edges is a separator. The *edge-connectivity* of G, denoted by $\kappa'(G)$, is the maximum integer k such that G is k-edge-connected.

For any vertex x, if S is a vertex-separator of $G-x$ then $S \cup \{x\}$ is a vertex-separator of G, hence

$$\kappa(G) \leq \kappa(G-x) + 1.$$

However, the connecivity of $G-x$ may not be upper bounded by a function of $\kappa(G)$; see Exercise 5.3 (ii).

Regarding egde-connectivity, things are a bit easier. Indeed, for any edge $e \in E(G)$, F is an edge-separator of $G \setminus e$ if and only if $F \cup \{e\}$ is an edge-separator of G. Hence

$$\kappa'(G) - 1 \leq \kappa'(G \setminus e) \leq \kappa'(G).$$

By definition, being 1-connected, 1-edge-connected or connected is equivalent. For larger value of k, k-connectivity implies k-edge-connectivity.

Proposition 5.1. *Let G be a graph with at least two vertices.*

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

Proof. Removing all edges incident to a vertex makes the graph disconnected. Hence, $\kappa'(G) \leq \delta(G)$.

Let us assume that $\kappa'(G) = k$. Let $F = \{x_1y_1, x_2y_2, \ldots, x_ky_k\}$ be an edge-separator of G such that all x_i's are in the same connected component C of $G \setminus F$. If $G - \{x_1, x_2, \ldots, x_k\}$ is not connected, then $\kappa(G) \leq k$. Else, $C = \{x_1, x_2, \ldots, x_k\}$. Hence, x_1 has at most k neighbours, namely the x_i's for $x_i \neq x_1$ and the y_i's such that $x_i = x_1$. Then the neighbourhood of x_1 is a vertex-separator of size at most k. So $\kappa(G) \leq k \leq \kappa'(G)$. □

Reciprocally, the edge-connectivity of a graph cannot be bounded by its connectivity. See Exercise 5.3.

A natural question is how to add a vertex (computer) to an already existing k-connected graph (network) so that it remains k-connected. Obviously, since the connectivity is at least the minimum degree by Proposition 5.1, one needs to link the new vertex to at least k existing vertices. This easy necessary condition is in fact sufficient.

Lemma 5.2. *Let G be a k-connected graph. If G' is obtained from G by adding a new vertex x adjacent to at least k vertices of G, then G' is k-connected.*

Proof. Let S be a separator S of G'. Let us show that $|S| \geq k$. If S contains x, then $S \setminus \{x\}$ must be a separator of G. Since G is k-connected then $|S \setminus \{x\}| \geq k$ and so $|S| \geq k+1$. Assume now that $x \notin S$. If $N(x) \subseteq S$ then $|S| \geq k$. Else, $N(x) \setminus S \neq \emptyset$ and $N(x) \setminus S$ belongs to a unique connected component of $G' \setminus S$ (the one of x). Hence, S is a separator of G. Thus $|S| \geq k$ because G is k-connected. □

Similarly, adding a new vertex of degree k to a k-edge-connected graph yields a k-edge-connected graph.

Lemma 5.3. *Let G be a k-edge-connected graph. If G' is obtained from G by adding a new vertex x adjacent to at least k vertices of G, then G' is k-edge-connected.*

Proof. Left in Exercise 5.5 □

5.2 2-edge-connected graphs

For 2-edge-connected graphs, there is a structural theorem similar to Theorem 1.15 for strongly connected digraphs. It can be proved in exactly the same way.

The following proposition follows easily from the definition of 2-edge-connectivity.

Proposition 5.4. *Let D be a 2-edge connected graph. Then every edge is in a cycle.*

Proof. Let $e = uv$ be an edge. Since G is 2-edge-connected then $G \setminus e$ is connected. Thus there is a (v,u)-path in $G \setminus e$. Its concatenation with (u,v) is a cycle containing e. □

Definition 5.5. Let G be a graph and H be a subgraph of G. A *H-handle* is a path or a cycle (all vertices are distinct except possibly the two endvertices) such that its endvertices are in $V(H)$ and its internal vertices are in $V(G) \setminus V(H)$. A *handle decomposition* of G is a sequence $(C, P_1, \ldots, P_k)$ such that:

- $C = G_0$ is a cycle;
- for all $1 \leq i \leq k$, P_i is a G_{i-1}-handle and $G_i = G_{i-1} \cup P_i$;
- $G_k = G$.

It is straightforward to show that if H is a 2-edge-connected subgraph of a graph G, the graph $H \cup P$ is 2-edge-connected for any H-handle P. (See Exercise 5.6.) Hence, an easy induction immediately yields that every graph admitting a handle decomposition is 2-edge-connected. Conversely, every 2-edge-connected graph admits a handle decomposition starting at any cycle.

Theorem 5.6. *Let G be a 2-edge-connected graph and C a cycle. Then G has a handle decomposition $(C, P_1, \ldots, P_k)$.*

Proof. Let H be the largest subgraph of G such that H admits a handle decomposition $(C, P_1, \ldots P_k)$. Since every edge xy in $E(G) \setminus E(H)$ with both endvertices in $V(H)$ is a H-handle, H is an induced subgraph of G. Suppose for a contradiction that $H \neq G$, then $V(H) \neq V(G)$. Since G is 2-edge connected, there is an edge vw with $v \in V(G)$ and $w \in V(G) \setminus V(H)$. Since G is 2-edge connected, G contains a (w,H)-path P. Then, the concatenation of (v,w) and P is a H-handle in G, contradicting the maximality of H. □

Corollary 5.7 (Robbins, 1939)**.** *A graph admits a strongly connected orientation if and only if it is 2-edge connected.*

Proof. Necessity: If a graph G is not connected, then there is no directed path between any two vertices in distinct components whatever be the orientation. Let us assume that G has an edge uv such that $G \setminus uv$ is not connected. Let C_u and C_v be the connected components of u and v in $G \setminus uv$. Then, if uv is oriented from u to v, there is no directed (v,u)-path using this orientation.

Sufficientcy: Let us assume that G is 2-edge connected. From Theorem 5.6, G admits a handle decomposition $(C, P_1, \ldots, P_k)$. Orienting C into a directed cycle and each P_i, $1 \leq i \leq k$, into a directed path, we obtain an orientation D of G having a handle decomposition. So, by Theorem 1.15, D is strongly connected. □

5.3 2-connected graphs

For 2-connected graphs, there is a structural theorem similar to Theorems 5.6 and 1.15.

Observe that since a 2-connected graph is also 2-edge-connected by Proposition 5.1, every edge of a 2-connected graph contains is in a cycle. More generally, for any two vertices x and y (not necessarily adjacent) there is a cycle containing x and y. See Exercise 5.7.

Definition 5.8. Let G be a graph and H be a subgraph of G. A *H-ear* is a path whose endvertices are in $V(H)$ and whose internal vertices are in $V(G) \setminus V(H)$. An *ear decomposition* of G is a sequence $(C, P_1, \ldots, P_k)$ such that:

- $C = G_0$ is a cycle;

- for all $1 \leq i \leq k$, P_i is a G_{i-1}-ear and $G_i = G_{i-1} \cup P_i$;
- $G_k = G$.

It is straightforward to show that if H is a 2-connected subgraph of a graph G, the graph $H \cup P$ is 2-connected for any H-ear P. (See Exercise 5.6.) Hence, an easy induction immediately yields that every graph admitting an ear decomposition is 2-connected. Conversely, every 2–connected graph admits an ear decomposition starting at any cycle.

Theorem 5.9. *Let G be a 2-connected graph and C a cycle. Then G has an ear decomposition $(C, P_1, \ldots, P_k)$.*

Proof. Let H be the largest subgraph of D such that H admits an ear decomposition $(C, P_1, \ldots P_k)$. Since every edge xy in $E(G) \setminus E(H)$ with both endvertices in $V(H)$ is a H-ear, H is an induced subgraph of G. Suppose for a contradiction that $H \neq G$, then $V(H) \neq V(G)$. Since G is connected, there is an edge vw with $v \in V(G)$ and $w \in V(G) \setminus V(H)$. Since G is 2-connected, $G - v$ contains a (w, H)-ear P. The concatenation of (v, w) and P is a H-ear in G, contradicting the maximality of H. □

5.4 Contraction and k-connected graphs

Definition 5.10. Let $e = xy$ be an edge of a graph $G = (V, E)$. Let G/e denote the graph obtained from G by *contracting* the edge e in a new vertex v_e that is adjacent to all neighbours of x and y. Formally, G/e has vertex set $V' = (V \setminus \{x, y\}) \cup \{v_e\}$ and edge set $E' = \{vw \in E \mid \{v, w\} \cap \{x, y\} = \emptyset\} \cup \{v_e w \mid w \in (N_G(x) \cup N_G(y)) \setminus \{x, y\}\}$.

If G is connected, then every edge e is contained in a spanning tree T (Exercice 1.22), and T/e is a spanning tree of G/e. Hence, we have the following.

Proposition 5.11. *Let G be a connected graph. For any edge $e \in E(G)$, the graph G/e is connected.*

This result cannot be extended to 2-connectivity: there are 2-connected graphs that have an edge e such that G/e is not 2-connected (See Exercice 5.9.). However, in a 2-connected graph, there is an edge the contraction of which results in a 2-connected graph.

Proposition 5.12. *If G is 2-connected and $|V(G)| > 3$, then there is an edge e of G such that G/e is 2-connected.*

Proof. Assume G is 2-connected and $|V(G)| > 3$. By Proposition 5.4, G contains a cycle. Let C be a cycle of minimum length. By Theorem 5.9, G admits an ear decomposition $(C, P_1, P_2, \ldots, P_r)$. Let m be the greatest index such that P_m is not an edge. Then, $G' = G \setminus \bigcup_{i=m+1}^{r} E(P_i)$ is a spanning 2-connected subgraph of G. Let e be an edge of P_m. If $m = 1$, then $G = C$ is a cycle of length at least 4 and thus G/e is a cycle and so 2-connected. If $m \geq 2$, then G'/e can be obtained from C by adding the H-paths $P_1, P_2, \ldots, , P_{m-1}, P_m/e$. Hence, from Theorem 5.9, G'/e is 2-connected. Since G/e is a supergraph of G'/e (obtained by adding the edges corresponding to P_i, $m < i \leq r$), then G/e is 2-connected. □

Lemma 5.13. *If G is 3-connected and $|V(G)| > 4$ then there is an edge e of G such that G/e is 3-connected.*

Proof. Assume for a contradiction that such an edge does not exist. Then, for any $xy \in E(G)$, the graph G/xy contains a separator S with at most two vertices. Since $\kappa(G) \geq 3$, the vertex v_{xy} of G/xy resulting from the contraction is in S and $|S| = 2$. Let z be the vertex in $S \setminus \{v_{xy}\}$. Then, $\{x,y,z\}$ is a separator of G. Since no proper subset of $\{x,y,z\}$ is a separator of G, every vertex in $\{x,y,z\}$ has a neighbour in each component of $G - \{x,y,z\}$.

Consider x, y and z, as above, such that a component C of $G - \{x,y,z\}$ has minimum size. Let v be the neighbour of z in C. By assumption, G/zv is not 3-connected, so there exists a vertex w such that $\{z,v,w\}$ is a separator of G. Again, every vertex in $\{z,v,w\}$ has a neighbour in every component of $G - \{z,v,w\}$.
At least one of x and y, say x is not w and so the connected component of x in $G - \{z,v,w\}$ contains all the connected components of $G - \{x,y,z\}$. Since x and y are adjacent, there is a component D of $G - \{z,v,w\}$ that contains none of x and y. Hence any other one (there is at least one since $G - \{z,v,w\}$ is not connected) must be included in C and thus is smaller than because $v \in C$. This contradicts the minimality of C.

□

Hence, any 3-connected graph can be reduced to K_4 by a succession of edge-contractions. One can show that, reciprocally, any such graph is 3-connected. See Exercise 5.22.

Theorem 5.14 (Tutte, 1961)**.** *A graph G is 3-connected if and only if there is a sequence $G_0,\ldots,G_n$ of graphs such that:*

(i) $G_0 = K_4$ and $G_n = G$;

(ii) for any $i < n$, there is an edge xy of G_{i+1} such that $d(x), d(y) \geq 3$ and $G_i = G_{i+1}/xy$.

Propositions 5.11 and 5.12 and Lemma 5.13 cannot be generalized to the 4-connected case. Indeed, the square of a cycle, depicted in Figure 5.1, is 4-connected. However, the contraction of any edge creates a vertex with degree three.

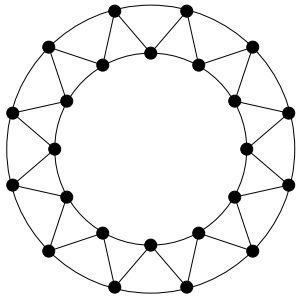


Figure 5.1: 4-connected graph such that the contraction of any edge makes it 3-connected

Any 4-connected graph (except K_5) can be modified into a smaller 4-connected graph by contracting one or two edges. If $k \geq 6$, for any $b \geq 1$, there are k-connected graphs with arbitrary

size that cannot be reduced to a k-connected graph by contracting at most b edges (e.g., planar triangulations of a torus). The question is open for $k = 5$. For more on this topic, we refer the reader to the survey of M. Kriesell [1].

5.5 Connectivity in digraphs

Similar concepts to k-connectivity and k-edge-connectivity may be defined for digraphs.

Let D be a digraph. Let W be an set of arcs (resp. of vertices). If $D \setminus W$ (resp. $D - W$) is not strongly connected, then W *separates* D, and W is called an *arc-separator* (resp. *vertex-separator*) or simply *separator* of D.

For any $k \geq 1$, D is *k-strongly connected* if if it has order at least $k+1$ and set of $k-1$ vertices is a separator. In particular, the complete symmetric digraph $\vec{K}_{k+1}$ is the only k-strongly connected digraph with $k+1$ vertices. The *strong connectivity* of D, denoted by $\kappa(D)$, is the maximum integer k such that D is k-strongly connected. Similarly, D is *k-arc connected* if it has at least two vertices and no set of $k-1$ arcs is a separator. The *arc-connectivity* of D, denoted by $\kappa'(D)$, is the maximum integer k such that D is k-arc-connected.

Results similar to the one proved in Section 5.1 hold for strong-connectivity and arc-connectivity. For any vertex x, if S is a vertex-separator of $D-x$ then $S \cup \{x\}$ is a vertex-separator of D, hence

$$\kappa(D) \leq \kappa(D-x) + 1.$$

However, the strong connectivity of $D-x$ may not be upper bounded by a function of $\kappa(D)$.

For any arc $e \in E(D)$, F is an arc-separator of $D \setminus e$ if and only if $F \cup \{e\}$ is an arc-separator of D. Hence

$$\kappa'(D) - 1 \leq \kappa'(D \setminus e) \leq \kappa'(D).$$

By definition, being 1-strongly connected, 1-arc-connected or strongly connected is equivalent. For larger value of k, k-strong connectivity implies k-arc-connectivity.

Proposition 5.15. *Let D be a graph with at least two vertices.*

$$\kappa(D) \leq \kappa'(D) \leq \min\{\delta^+(D), \delta^-(D)\}.$$

Proof. Removing all arcs leaving a vertex results in a digraph which not strongly connected. Hence, $\kappa'(D) \leq \delta^+(D)$. By directional duality, $\kappa'(D) \leq \delta^+(D)$.

Let us assume that $\kappa'(D) = k \geq 2$. Let $F = \{x_1y_1, x_2y_2, \ldots, x_ky_k\}$ be an arc-separator of D. Then $D \setminus F$ has two strongly connected components X and Y such that $x_i \in X$ and $y_i \in Y$, for all $1 \leq i \leq k$ and there is no arc with tail in X and head in Y except those of F (see Exercise 5.4-2)). If $D-$ is not strongly connected, then $\kappa(G) \leq k$. Else $X = \{x_1, \ldots, x_k\}$. Then x_1 has at most k outneighbours, namely the x_i's for $x_i \neq x_1$ and the y_i's such that $x_i = x_1$. Then the outneighbourhood of x_1 is a vertex-separator of size at most k. So $\kappa(D) \leq k \leq \kappa'(D)$. □

Reciprocally, the arc-connectivity of a graph cannot be bounded by its strong connectivity.

In fact, as we shall see, all the results on connectivity and edge-connectivity may be seen as particular cases of results on strong connectivity and arc-connectivity for symmetric digraphs. Indeed, for any graph G, its *associated digraph*, denoted $\vec{G}$ is the symmetric digraph obtained from G by replacing each edge xy by the two arcs (x,y) and (y,x).

The following proposition follows directly form the definition of $\vec{G}$.

Proposition 5.16. *Let G be a graph and $\vec{G}$ is associated digraph.*

(i) *$(v_1,v_2,\ldots,v_p)$ is a path in G if and only if $(v_1,v_2,\ldots,v_p)$ is a directed path in $\vec{G}$.*

(ii) *G is connected if and only if $\vec{G}$ is strongly connected.*

Theorem 5.17. *Let G be a graph and $\vec{G}$ is associated digraph. Then*

(i) $\kappa(G) = \kappa(\vec{G})$.

(ii) $\kappa'(G) = \kappa'(\vec{G})$.

Proof. Let W be a set of vertices. Then the digraph associated to $G-W$ is $\vec{G}-W$. Hence by Proposition 5.16-(ii) the graph $G-W$ is connected if and only if $\vec{G}-W$ is strongly connected. In other words, W is a vertex-separator of G if and only if it is a vertex-separator of $\vec{(G)}$. This implies (i).

Let us now prove (ii). Let F be a minimum edge-separator of G. Then $G \setminus F$ has two components X and Y (see Exercise 5.4-1)). Let $F = \{x_1y_1, x_2y_2, \ldots, x_ky_k\}$ with $x_i \in X$ and $y_i \in Y$, for all $1 \leq i \leq k$. Then $\vec{F} = \{(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k)\}$ is an arc-separator of $\vec{G}$ since there are no arcs with tail in X and head in Y. So $\kappa'(\vec{G}) \leq \kappa'(G)$.

Conversely assume that $\vec{F} = \{(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k)\}$ is a minimum arc-separator of $\vec{G}$. Then $\vec{G} \setminus \vec{F}$ has two strongly connected components X and Y such that $x_i \in X$ and $y_i \in Y$, for all $1 \leq i \leq k$ and there is no arc with tail in X and head in Y except those of $\vec{F}$ (see Exercise 5.4-2)). Thus $F = \{x_1y_1, x_2y_2, \ldots, x_ky_k\}$ is an edge-separator of G. □

In view of Theorem 5.17, Proposition 5.1 may be seen as Proposition 5.15 in the case of symmetric digraphs.

5.6 Menger's Theorem

Let W be an edge-separator (resp. vertex-separator) of a graph G. If two vertices u and v are in two distinct connected components of $G \setminus W$ (resp. $G-W$) , then W separates two vertices u and v and is called a (u,v)-*separator*. A separator of a graph is necessarily a (u,v)-separator for some pair of vertices. In addition, if we consider a vertex-separator, these two vertices are not adjacent.

Let u and v be two vertices of a graph G. The *edge-connectivity between u and v* or (u,v)-*edge-connectivity* in G, denoted by $\kappa'_G(u,v)$ or simply $\kappa'(u,v)$, is the minimum cardinality of a (u,v)-edge-separator. If u and v are not adjacent, then the *connectivity between u and v* or (u,v)-*connectivity* in G, denoted by $\kappa_G(u,v)$ or simply $\kappa(u,v)$, the minimum cardinality of a

(u,v)-vertex-separator. Clearly, $\kappa(G) = \min\{\kappa(u,v) \mid u,v \in V(G), uv \notin E(G)\}$ and $\kappa'(G) = \min\{\kappa'(u,v) \mid u,v \in V(G)\}$. So, to compute $\kappa(G)$ (resp. $\kappa'(G)$), it is sufficient to compute $\kappa(u,v)$ (resp. $\kappa'(u,v)$) for every pair of vertices u and v.

Similar concept may be defined in digraphs. Let W be an arc-separator (resp. vertex-separator) of a digraph D. If there is no directed (u,v)-path in $D \setminus W$ (resp. $D - W$) , then W separates u from v and is called a (u,v)-*separator*. Observe that contrary to the undirected case, a (u,v)-separator is not necessarily a (v,u)-separator. A separator of a digraph is necessarily a (u,v)-separator for some pair of vertices. In addition, if we consider a vertex-separator, uv is not an arc (but vu may be an arc).

Let u and v be two vertices of a digraph D. The *arc-connectivity between u and v* or (u,v)-*arc-connectivity* in D, denoted by $\kappa'_D(u,v)$ or simply $\kappa'(u,v)$, is the minimum cardinality of a (u,v)-edge-separator. If uv is not an arc, then the *connectivity between u and v* or (u,v)-*connectivity* in D, denoted by $\kappa_D(u,v)$ or simply $\kappa(u,v)$ the minimum cardinality of a (u,v)-vertex-separator. Clearly, $\kappa(G) = \min\{\kappa(u,v) \mid u,v \in V(G), uv \notin E(G)\}$ and $\kappa'(G) = \min\{\kappa'(u,v) \mid u,v \in V(G)\}$. So, to compute $\kappa(G)$ (resp. $\kappa'(G)$), it is sufficient to compute $\kappa(u,v)$ (resp. $\kappa'(u,v)$) for every pair of vertices u and v.

Two (directed) paths are *independent* if their internal vertices are distinct. In particular, two (directed) (s,t)-paths are independent if their sole common vertices are s and t. The maximum number of pairwise independent (directed) (s,t)-paths is denoted by $\Pi(s,t)$. If W is an (s,t)-vertex-separator of a graph or digraph, then two independent (s,t)-paths intersect W in distinct vertices, so

$$\kappa(s,t) \leq \Pi(s,t). \tag{5.1}$$

Similarly, if F is an (s,t)-edge-separator in a graph, then two edge-disjoint (s,t)-paths intersect W in distinct edges, and if F is an (s,t)-arc separator in a digraph, then two arc-disjoint directed (s,t)-paths intersect W in distinct arcs. Hence denoting by $\Pi'(s,t)$ the maximum number of pairwise edge-disjoint (s,t)-paths (resp. arc-disjoint directed (s,t)-paths), we have

$$\kappa'(s,t) \leq \Pi'(s,t). \tag{5.2}$$

Menger's Theorem shows that Inequalities 5.1 and 5.2 are in fact equalities.

Theorem 5.18 (Menger, 1927)**.** *(i) Let s and t be two distinct vertices of a graph (resp. digraph) such that st is not an edge (resp. arc). Then, the minimum size of an (s,t)-vertex-separator equals the maximum number of independent (s,t)-paths. In symbols,*

$$\kappa(s,t) = \Pi(s,t).$$

(ii) Let s and t be two distinct vertices of a graph (resp. a digraph). Then, the minimum size of an (s,t)-edge-separator (resp. arc-separator) equals the maximum number of pairwise edge-disjoint (s,t)-paths (resp. pairwise arc-disjoint directed (s,t)-paths). In symbols,

$$\kappa'(s,t) = \Pi'(s,t).$$

Proof. Let us first prove (ii) in a digraph D. We will use a recursive algorithmic approach as follows. Suppose we have found a set of k pairwise arc-disjoint directed (s,t)-paths, then we will either construct a set of $k+1$ pairwise arc disjoint paths from s to t or find an (s,t)-arc-separator of size k.

The induction can be started with $k=0$ or $k=1$ by finding a directed (s,t)-path.

Let $\mathcal{P}=\{P_1,\ldots,P_k\}$ be a set of k pairwise arc-disjoint directed (s,t)-paths. Let $E(\mathcal{P})=E(P_1)\cup\cdots\cup E(P_k)$. Let us construct a set S of vertices with the following algorithm.

Algorithm 5.1 (Constructing S)**.**

1. Put s in S.
2. If there exist $x\in S$ and an arc xy in $E(D)\setminus E(\mathcal{P})$ (i.e. xy is in none of the paths P_i), then add y to S. Go to 2.
3. If there exist $x\in S$ and an arc yx in $E(\mathcal{P})$, then add y to S. Go to 2.

Observe that the so constructed set S is connected but not necessarily strongly connected. (The graph underlying $D\langle S\rangle$ is connected). Two cases can appear at the end of the algorithm.
Case 1: $t\in S$. In that case we will construct a set of $k+1$ pairwise arc-disjoint directed (s,t)-paths.

Since S is connected there is an *oriented* (not necessarily directed) (s,t)-path P_{k+1}, that is a sequence $x_0e_1x_1\ldots x_je_{j+1}x_{j+1}\ldots e_px_p$ where $x_0=s$, $x_p=t$ and for all $1\leq j\leq p$, e_j is either the arc $x_{j-1}x_j$ or the arc x_jx_{j-1}. If $e_j=x_{j-1}x_j$, then e_j is called a *forward arc*; if not it is called a *backward arc*. Observe that by construction of S, if an arc is forward in P_{k+1} if and only if it is not in $E(\mathcal{P})$.

If P_{k+1} does not contain any backward arc, then the set of paths $(P_1,P_2,\ldots,P_k,P_{k+1})$ is a set of $k+1$ pairwise arc-disjoint directed (s,t)-paths.

Otherwise starting from the set $\mathcal{P}$ and P_{k+1}), we shall construct a set $\mathcal{P}'=(P'_1,\ldots,P'_k)$ of k pairwise arc-disjoint directed (s,t)-paths and an oriented (s,t)-path P'_{k+1} having one backward arc fewer than P_{k+1} and such that if an arc is forward in P'_{k+1} if and only if it is not in $E(\mathcal{P}')$. Repeating this construction p times (where$p$ is the number of backward arcs in P_{k+1}), we obtain a set of $k+1$ pairwise arc-disjoint directed (s,t)-paths.

Let j be the smallest index for which e_j is backward and let i_0 be the index such that $e_j\in E(P_{i_0})$. Let us define the following paths.

- $P'_i=P_i$ for $1\leq i\neq i_0\leq k$;

  P'_{i_0} is the concatenation of the directed (s,x_{j-1})-subpath of P_{k+1} and the directed (x_{j-1},t)-subpath of P_{i_0};

  P'_{k+1} is the concatenation of the directed (s,x_j)-subpath of P_{i_0} and the oriented (x_j,t)-subpath of P_{k+1}.

It is simple matter to check that the paths $P'_1,\ldots,P'_k,P'_{k+1}$ satisfy the property describe above.

Case 2: $t \notin S$. In that case we will find an (s,t)-arc-separator of size k.

Let $T = V \setminus S$. Let F be the set of arcs from S to T, i.e. with tail in S and head in T. Then F separates S from T and so form an (s,t)-arc-separator. But every such arc belongs to $E(\mathcal{P})$; otherwise we could have applied Step 2 of Algorithm 5.1 and so this algorithm was not finished. Furthermore, a path P_i cannot contain two arcs from S from T, otherwise there will exist an arc yx from T to S in P_i and this arc should have been added to the set S by Step 3 of Algorithm 5.1. Hence $|F| \leq k$.

Let us deduce (i) for digraphs. We outline the proof. Some details are left in Exercise 5.16.

Let D be a digraph and s and t two vertices such that st is not an arc. The *split digraph* $S(D)$ is the digraph D obtained in splitting every vertex into an arc v^-v^+:

$$\begin{aligned} V(S(D)) &= \bigcup_{v\in V(D)} \{v^-,v^+\}, \\ E(S(D)) &= \{v^-v^+ \mid v \in V(D)\} \cup \{u^+v^- \mid uv \in E(D)\}. \end{aligned}$$

An arc of the form v^-v^+ is called an *inner arc* of $S(D)$.

Trivially, if W is an (s,t)-vertex-separator of D then $\{v^-v^+ \mid v \in W\}$ is an (s^+,t^-)-arc-separator of $S(D)$. Conversely, one can show that there is a minimum (s^+,t^-)-arc-separator F in $S(D)$ made of inner arcs and for such an F, the set $\{v \mid v^-v^+ \in F\}$ is an (s,t)-vertex-separator of D. Hence $\kappa_D(s,t) = \kappa'_{S(D)}(s^+,t^-)$.

For any directed path $P = (x_1,\ldots,x_p)$, its *split path* $S(P)$ is defined as the directed path $(x_1^+,x_2^-,x_2^+,\ldots,x_{p-1}^+,x_p^-)$. One can easily see that every directed (s^+,t^-)-path is the split path of some directed (s,t)-path. Moreover, one shows that $P_1,\ldots,P_k$ are pairwise independent directed (s,t)-path in D if and only if $S(P_1),\ldots,S(P_k)$ are pairwise arc-disjoint directed (s^+,t^-)-path in $S(D)$. Hence, $\Pi_D(s,t) = \Pi'_{S(D)}(s^+,t^-)$.

Now by (ii) for digraphs, we have $\kappa'_{S(D)}(s^+,t^-) = \Pi'_{S(D)}(s^+,t^-)$. Thus $\kappa_D(s,t) = \Pi_D(s,t)$.

Both assertions (i) and (ii) for graphs may be deduced from themselves for digraphs using the fact that connectivity in graphs corresponds to connectivity in symmetric digraphs. Let G be a graph and $\vec{G}$ its associated digraph. Similarly to Theorem 5.17, one can show that $\kappa_G(s,t) = \kappa_{\vec{G}}(s,t)$ and $\kappa'_G(s,t) = \kappa'_{\vec{G}}(s,t)$. It is also not difficult to show that $\Pi_G(s,t) = \Pi_{\vec{G}}(s,t)$ and $\Pi'_G(s,t) = \Pi'_{\vec{G}}(s,t)$. (See Exercise 5.15). □

We now give a short inductive proof of Theorem 5.18-(i) for graphs.

Alternative proof of Theorem 5.18-(i) for graphs: For sake of contradiction, let $k = \kappa(s,t)$ be the smallest integer contradicting the theorem. Clearly, $k \geq 2$. Let G be a counterexample (for this minimum value of k) that has the minimum number of edges. Then, there are at most $k-1$ independent (s,t)-paths.

There is no vertex x adjacent both to s and t otherwise $G-x$ would be a counterexample for $k-1$, a contradiction to the minimality of k.

Let W be an (s,t)-vertex-separator of size k.

Let us assume first that both s and t are not adjacent to all vertices in W. Then, each of s and t has a neighbour in $V(G) \setminus W$ (otherwise, the neighbourhood of s or t would be a smallest vertex-separator). Let G_s be the graph obtained from G by contracting the component C of $G \setminus W$ containing s in a single vertex s' (i.e. replacing C by a single vertex adjacent to all vertices in W). In G_s, an (s',t)-vertex-separator has size at least k. Since C has at least 2 vertices, G_s has less edges than G. By minimality of G, there are at least k independent (s',t)-paths in G_s. Removing s', we obtain k paths from W to t such that, for any $w \in W$, w is the start of exactly one of these paths. Performing the same operation in G_t (obtained in the same way as G_s), we obtain k independent paths from s to W such that, for any $w \in W$, w is the terminus of exactly one of these paths. For any $w \in W$, let us concatenate the (s,w)-path and the (w,t)-path. We obtain k independent (s,t)-paths in G, a contradiction.

So, we can assume that, for every (s,t)-vertex-separator W of size k, either s or t is adjacent to all vertices of W. Let $P = (s,x_1,x_2,\ldots,x_l,t)$ be a shortest (s,t)-path. Then, $l \geq 2$b because s and t have no common neighbours. By minimality of G, in $G \setminus x_1x_2$, there is an (s,t)-vertex-separator W_0 of size $k-1$. Hence, $W_1 = W_0 \cup \{x_1\}$ and $W_2 = W_0 \cup \{x_2\}$ are (s,t)-vertex-separators in G. Since s is not adjacent to x_2 because P is a shortest path, then t is adjacent to all vertices of W_2. Similarly, s is adjacent to all the vertices in W_1. Hence, all vertices of W_0 (which is not empty) are common neighbours of s and t, a contradiction.

□

Corollary 5.19 (Menger, 1927)**.** *Let G be a graph with at least two vertices.*

(i) G is k-connected if and only if any two vertices can be joined by k independent paths.

(ii) G is k-edge-connected if and only if any two vertices can be joined by k edge-disjoint paths.

Proof. (i) If any two vertices can be joined by k independent paths, then G is clearly k-connected. Now, if G is k-connected, by Theorem 5.18-(i), any 2 non-adjacent vertices can be joined by k independent paths. It remains to show that if G is k-connected, then any two adjacent vertices x and y can be joined by k independent paths.

Let G' be obtained from G by adding a vertex x' adjacent to all neighbours of x and a vertex y' adjacent to all neighbours of y. Since $\delta(G) \geq \kappa(G) \geq k$, by Lemma 5.2, G' is k-connected. Since x' and y' are not adjacent, there are k independent (x',y')-paths $P_1,\ldots,P_k$ in G'. For any $1 \leq i \leq k$, let P'_i be the path obtained as follows:

- if $\{x,y\} \subset V(P_i)$ then $P'_i = (x,y)$;
- if $\{x,y\} \cap V(P_i) = \{x\}$, take the (x,y')-subpath of P_i and replace y' by y;
- if $\{x,y\} \cap V(P_i) = \{y\}$, take the (x',y)-subpath of P_i and replace x' by x;
- if $\{x,y\} \cap V(P_i) = \emptyset$, P'_i is obtained by replacing x' with x and y' with y.

We get k independent (x,y)-paths.

(ii) Straightforward from Theorem 5.18-(ii). □

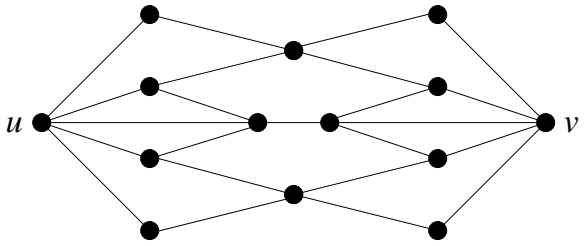
Corollary 5.20. *Let G be a k-connected graph and let A and B be two subsets of $V(G)$. If $|A| \geq k$ and $|B| \geq k$, then there are k disjoint (A,B)-paths*

Proof. Let G' be the graph obtained from G by adding two vertices a and b respectively adjacent to all vertices of A and B. From Lemma 5.2 (applied twice), G' is k-connected and so $\kappa_{G'}(a,b) \geq k$. By Menger's Theorem, there are k independent (a,b)-paths in G'. Removing a and b from these paths, we obtain k disjoint (A,B)-paths.

□

5.7 Exercises

Exercise 5.1. Compute $\kappa(u,v)$ and $\kappa'(u,v)$ in the graph below:



Exercise 5.2. Prove the following assertion or give a counterexample. *If P is a (u,v)-path in a 2-connected graph G, then there exists a (u,v)-path Q independent of P.*

Exercise 5.3. Let k and l be two integers with $1 \leq k < l$. Give graphs G_1, G_2 and G_3 such that :

- (i) $\kappa(G_1) = 1$ and $\kappa'(G_1) = l$;
- (ii) $\kappa(G_2) = k$ and $\kappa(G_2 - x) = l$ for some particular vertex x;
- (iii) $\kappa'(G_3 - x) = k$ and $\kappa'(G_3 \setminus xy) = l$ for some particular edge xy.

Exercise 5.4. 1) Show that if the edge-connectivity of a graph is $k \geq 1$, then when removing at most k edges then we obtain at most two connected components. Does there exists a similar result for connectivity? arc-connectivity?

2) Let D be a k-arc connected digraph and $F = \{x_1y_1, \ldots, x_ky_k\}$ a minimum arc-separator. Show that $D \setminus F$ has two strongly connected components X and Y such that $x_i \in X$ and $y_i \in Y$, for all $1 \leq i \leq k$ and there is no arc with tail in X and head in Y except those of F.

Exercise 5.5. Prove Lemma 5.3.

Exercise 5.6. Show that if H is a 2-edge-connected subgraph of a graph G, then for any H-handle P, the graph $H \cup P$ is 2-edge-connected.

Exercise 5.7. Let G be a graph on at least 2 vertices. Show that the following propositions are equivalent:

- (i) G is 2-connected;
- (ii) any two vertices are in a cycle;
- (iii) any two edges are in a cycle and $\delta(G) \geq 2$;
- (iv) for any three vertices x,y et z, there is a (x,z)-path containing y.

Exercise 5.8. Let G be a graph on at least 3 vertices. Show that the following propositions are equivalent:

- (i) G is 2-edge-connected;
- (ii) any edge is in a cycle;
- (iii) any two edges are in a tour and $\delta \geq 1$;
- (iv) any two vertices are in a tour.

Exercise 5.9. Give an example of a 2-connected graph G with an edge e such that G/e is not 2-connected.

Exercise 5.10. Let $d_1 \leq d_2 \leq \cdots \leq d_n$ be the degree sequence of a graph. We assume that $d_j \geq j+k-1$ for $1 \leq j \leq n-1-d_{n-k+1}$.
Show that G is k-connected.

Exercise 5.11. Let G be a regular bipartite graph on at least two vertices. Prove that $\kappa(G) \neq 1$.

Exercise 5.12. Show a graph which is not 2-connected but admits a strongly connected orientation.

Exercise 5.13. Inspired by Algorithm 2.6, give an algorithm in time $O(|E|)$ that computes the 2-connected components of a graph.

Exercise 5.14. Let G be a connected graph, all vertices of which have even degree. Show that G is 2-edge-connected.

Exercise 5.15. Let G be a graph and $\vec{G}$ its associated digraph. Show that

(a) there are k independent (s,t)-paths in G if and only there are k independent directed (s,t)-paths in $\vec{G}$, and

(b) there are k pairwise edge-disjoint (s,t)-paths in G if and only there are k pairwise arc-disjoint directed (s,t)-paths in $\vec{G}$.

Exercise 5.16. Let D be a digraph and $S(D)$ its split digraph. Let s and t be two vertices of D such that st is not an arc.

1) a) Let F be an (s^+,t^-)-arc-separator in $S(D)$. For an non-inner arc $e = u^+v^-$, we define $r(e)$ to be u^-u^+ if $u \neq s$ and v^-v^+ otherwise. Show that if e is non-inner then the set $(F \setminus \{e\}) \cup \{r(e)\}$ is also an (s^+,t^-)-arc-separator.

b) Show that if F is an (s^+,t^-)-arc-separator made of inner arcs, then $\{v \mid v^-v^+ \in F\}$ is a vertex-separator of $S(D)$.

c) Deduce $\kappa_D(s,t) = \kappa'_{S(D)}(s^+,t^-)$.

2) a) Shows that $P_1,\dots,P_k$ are pairwise independent directed (s,t)-paths in D if and only if $S(P_1),\dots,S(P_k)$ are pairwise arc-disjoint directed (s^+,t^-)-paths in $S(D)$. Hence,

b) Deduce $\Pi_D(s,t) = \Pi'_{S(D)}(s^+,t^-)$.

Exercise 5.17.
Let G be a graph on at least three vertices which is not a complete graph.
1) Show that G has three vertices u, v, and w such that $uv \in E(G)$, $vw \in E(G)$ and $uw \notin E(G)$.
2) Show that if G is 2-connected and $\delta(G) \geq 3$ then there exists such a triple u,v,w such that, in addition, $G-\{u,w\}$ is connected.

Exercise 5.18. Let a and b be two vertices of a graph G. Let X and X' be two (a,b)-vertex-separators. Let us denote C_a (resp. C'_a) the connected component of a in $G-X$ (resp. $G-X'$) and C_b (resp. C'_b) the connected component of b in $G-X$ (resp. $G-X'$).

Prove that the two sets $Y_a = (X \cap C'_a) \cup (X \cap X') \cup (X' \cap C_a)$ and $Y_b = (X \cap C'_b) \cup (X \cap X') \cup (X' \cap C_b)$ are (a,b)-separators.

Exercise 5.19 (Dirac, 1960)**.** Let x be a vertex of a graph G and U a set of vertices of G not containing x. An (x,U)-*fan* is a set of (x,U)-paths such that the intersection of any two is $\{x\}$. Prove that G is k-connected if and only if it has at least $k+1$ vertices and for any choice of x and U such that $x \notin U$ and $|U| \geq k$, there is a (x,U)-fan of cardinality k.

Exercise 5.20. Let $k \geq 2$ be an integer. Prove that, if G is k-connected, then any set of k vertices is contained in a cycle. Is the converse also true?

Exercise 5.21. Let G be a cubic graph.
1) Show that if $\kappa'(G) \geq 2$ then $\kappa(G) \geq 2$.
2) Show that if $\kappa'(G) = 3$ and $\kappa(G) \geq 2$ then $\kappa(G) = 3$.

Exercise 5.22. Let x and y be two adjacent vertices of degree at least k in a graph G. Show that if G/xy is k-connected then G is also k-connected.

Exercise 5.23. Let $k \geq 2$ be an integer. Let G be a k-connected graph and xy an edge of G. Show that G/xy is k-connected if and only if $G-\{x,y\}$ is $(k-1)$-connected.

Exercise 5.24. Let G be a 2-connected graph of order at least 4. Prove that for every edge e, $G \setminus e$ or G/e is 2-connected.

Exercise 5.25. Let v be a vertex of a 2-connected graph G. Show that v has a neighbour u such that $G-\{u,v\}$ is connected.

Exercise 5.26. Let xy be an edge of a 2-connected graph G. Show that $G \setminus xy$ is 2-connected if and only if x and y are in a cycle of $G \setminus xy$.

Exercise 5.27. A graph G is *non-separable* if $G = K_2$ or G is 2-connected. A *block* of a graph is a subgraph which is non-separable and is maximal with respect to this property.

1) Show that two blocks intersect in at most one vertex.

2) Show that if a vertex v is in two blocks if and only if it is a *separating vertex*, that is $\{v\}$ is a separator.

3) The *block graph* of G, denoted $B(G)$, is the graph whose vertices are the blocks of G and in which two blocks are joined by an edge if they intersect. Show that if G is connected, then $B(G)$ is a tree.

4) The *end-blocks* of G are the blocks which are leaves in $B(G)$. Assume that B_1 and B_2 are two different end-blocks of a connected graph G. Show that if v_1 and v_2 are two vertices of B_1 and B_2 respectively which are not separating, then $G - \{v_1, v_2\}$ is connected.

Exercise 5.28 (W. T. Tutte)**.** A *wheel* is a graph obtained from a cycle by adding a vertex adjacent to all vertices of the cycle.

Let G be a 3-connected graph different from a wheel. Show that, for any edge e, either G/e or $G \setminus e$ is also a 3-connected graph.

Bibliography

[1] M. Kriesell. A survey on contractible edges in graphs of a prescribed vertex connectivity. *Graphs and Combinatorics* 18(1):1–30, 2002.

Chapter 6

Matching in Graphs

Let G be a graph. Two edges are *independent* if they have no common endvertex. A set M of independent edges of G is called a *matching*. The *matching number*, denoted $\mu(G)$, is the maximum size of a matching in G.

In this chapter, we consider the problem of finding a *maximum matching*, i.e. with maximum size. In particular, we will try to characterise the graphs G that admit a *perfect matching*, i.e. a matching covering all vertices of G.

Let M be a matching. The vertices that are incident to an edge of M are *matched* or *covered* by M. If U is a set of vertices covered by M, then we say that M *saturates* U. The vertices which are not covered are said to be *exposed*.

Let $G = (V,E)$ be a graph and M a matching. An *M-alternating* path in G is a path whose edges are alternatively in $E \setminus M$ and in M. An M-alternating path whose two endvertices are exposed is *M-augmenting*. We can use an M-augmenting path P to transform M into a greater matching (see Figure 6.1). Indeed, if P is M-alternating, then the symmetric difference between M and $E(P)$

$$M' = M \triangle E(P) = (M \setminus (E(P) \cap M) \cup (E(P) \setminus M)$$

is also a matching. Its size $|M'|$ equals $|M| - 1 + x$ where x is the number of exposed ends of P.

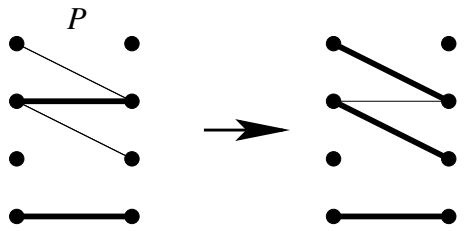


Figure 6.1: Getting of a greater matching from an augmenting path P. Bold edges are the one of the matching.

Hence if M is maximum, there are no augmenting paths. In fact, as shown by Berge, this necessary condition is also sufficient.

Theorem 6.1 (Berge 1957)**.** *Let M be a matching in a graph G. Then M is maximum if and only if there are no M-augmenting paths.*

Proof. Necessity was shown above so we just need to prove sufficiency. Let us assume that M is not maximum and let M' be a maximum matching. The symmetric difference $Q = M \triangle M'$ is a subgraph with maximum degree 2. Its connected components are cycles and paths where the edges of M and M' alternate. Hence, the cycles have even length and contain as many edges of M and of M'. Since M' is greater than M, Q contains at least one path P that contains more edges of M' than of M. Therefore, the first and the last edges of P belong to M', and so P is M-augmenting.

□

6.1 Matching in bipartite graphs

Let $G = ((A,B),E)$ be a bipartite graph. If $|A| \leq |B|$, the size of maximum matching is at most $|A|$. We want to decide whether it exists a matching saturating A. If there is such a matching M, then, for any subset S of A, the edges of M link the vertices of S to as many vertices of B.

Hence, we have a necessary condition, known as *Hall's c Condition*, for the existence of a matching saturating A:

$$|N(S)| \geq |S| \qquad \text{for all } S \subseteq A \tag{6.1}$$

where $N(S)$ is the set of vertices of $G \setminus S$ adjacent to at least one vertex of S. The set $N(S)$ is called the *neighbourhood* of S: $N(S) = \bigcup_{s \in S} N(s) \setminus S$.

Actually, this condition is sufficient.

Theorem 6.2 (Hall 1935)**.** *Let $G = ((A,B),E)$ be a bipartite graph. G has a matching saturating A if and only if $|N(S)| \geq |S|$ for all $S \subseteq A$.*

We give two proofs of this theorem. The first one uses some basic arguments, while the second one is based on augmenting paths.

First proof: By induction on $|A|$, the result holding trivially for $|A| = 1$. Let us assume that $|A| \geq 2$ and that Condition (6.1) is sufficient for any matching saturating A' with $|A'| < |A|$.

i) Assume first that $|N(S)| \geq |S| + 1$ for every non-empty proper subset S of A. Let $ab \in E(G)$ and consider $G' = G - \{a,b\}$. Then, any subset $S \subseteq A \setminus \{a\}$ satisfies:

$$|N_{G'}(S)| \geq |N_G(S)| - 1 \geq |S|.$$

By the induction hypothesis, G' has a matching M' saturating $A \setminus \{a\}$. Hence, the matching $M' \cup \{ab\}$ is a matching of G saturating A.

ii) Assume now that there is a non-empty proper subset A' of A such that $|N(A')| = |A'|$. By the induction hypothesis, the subgraph G' induced by $A' \cup N(A')$ admits a matching M'

saturating A'.
Let $G'' = G - G'$. For any set $S \subseteq A \setminus A'$,

$$|N_{G''}(S)| \geq |N_G(S \cup A')| - |N_G(A')| \geq |S \cup A'| - |A'| \geq |S|.$$

Hence, by the induction hypothesis, G'' admits a matching M'' saturating $A \setminus A'$. The union $M' \cup M''$ is a matching of G saturating A. □

Second proof: The proof is algorithmic. Given a matching M with maximum size which does not cover a_0, it returns a set $S \subseteq A$ such that $|N(S)| < |S|$. Let $A_0 = \{a_0\}$ and $B_0 = N(a_0)$. Note that all vertices of B_0 are covered (if $b_0 \in B_0$ is not covered, the edge a_0b_0 can be added to the matching). If $B_0 = \emptyset$, $S = A_0$ is a set such that $|N(S)| < |S|$ and the algorithm terminates. Else, B_0 is matched with $|B_0|$ vertices of A distinct from a_0. We set $A_1 = N_M(B_0) \cup \{a_0\}$, where $N_M(B_0)$ is the set of vertices matched with vertices of B_0. We have $|A_1| = |B_0| + 1 \geq |A_0| + 1$. Let $B_1 = N(A_1)$. Again, no vertices in B_1 is exposed, otherwise there is an M-augmenting path. If $|B_1| < |A_1|$, the algorithm terminates with $|N(A_1)| < |A_1|$. If not, let $A_2 = N_M(B_1) \cup \{a_0\}$. Then$|A_2| \geq |B_1| + 1 \geq |A_1| + 1$. And so on, the following algorithm is executed until it terminates.

Algorithm 6.1 (Finding a set violating Hall's Condition)**.**

1. $A_0 := \{a_0\}$, $i := 0$;
2. If $|B_i| = |N(A_i)| < |A_i|$, terminate and return A_i;
3. Else, do $A_{i+1} := A_i \cup N_M(B_i)$, $i := i + 1$ and go to Step 2.

The algorithm eventually terminates because the sequence $|A_i|$ is strictly increasing. Hence, it returns a set $S \subseteq A$ such that $|N(S)| < |S|$. □

Hall's Condition (6.1) applies for matching saturating A but it can be generalised for matching of any size.

Theorem 6.3. *Let $G = ((A,B),E)$ be a bipartite graph and $k \in \mathbb{N}$. G has a matching of size k if and only if $|N(S)| \geq |S| - |A| + k$ for any $S \subseteq A$.*

Proof. Let us add $|A| - k$ new vertices to B, each of them being linked to all vertices of A. Then, in this new graph, $|N(S)| \geq |S|$ for any $S \subset A$. Thus, by Hall's Theorem (6.2), the new graph has a matching saturating A. But at most $|A| - k$ edges (the ones incident to a new vertex) are not in G. Hence G has a matching of size at least k. □

Corollary 6.4. *If $G = ((A,B),E)$ is a k-regular bipartite graph ($k \geq 1$), then G has a perfect matching.*

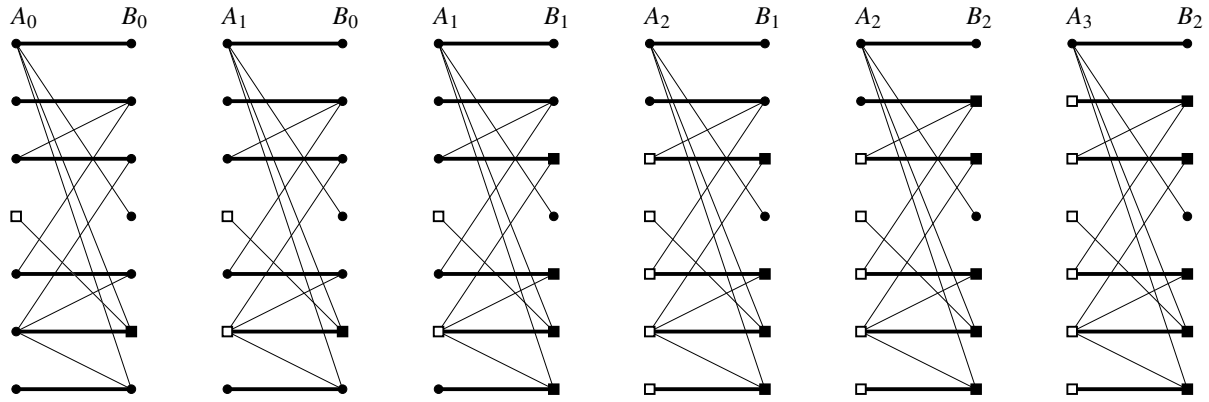


Figure 6.2: A run of Algorithm 6.1. The bold edges are those of the maximum matching. The vertices of A_i (resp. B_i) are represented by white (resp. black) squares.

Proof. If G is k-regular, then clearly $|A| = |B|$. So every matching saturating A is perfect.

Let $S \subseteq A$. The set S is incident to $k|S|$ edges that belong to the $k|N(S)|$ edges incident to $N(S)$. Hence, $k|S| \leq k|N(S)|$. Thus, G satisfies Hall's Condition (6.1) and so by Theorem 6.2, admits a matching saturating A, which is a perfect matching. □

Using the same method as in the second proof of Hall's Theorem, we give an algorithm which, given a bipartite graph $((A,B),E)$ computes either a matching saturating A or a set S such that $|N(S)| < |S|$. This algorithm, known as the *hungarian method*, is based on the sequence $a_o, b_1, a_1, b_2, a_2, \ldots$ viewed as a tree T.

Algorithm 6.2 (Hungarian Method)**.**

0. Start with any matching M.
1. If M saturates A, then return M. Else, let $a_0 \in A$ be an exposed vertex. Let T be the tree consisting of the single vertex a_0. Let $A' = V(T) \cap A$ and $B' = V(T) \cap B$.
2. If $N(A') = B'$ then $|N(A')| < |A'|$ because $|A'| = |B'| + 1$. Return $S = A'$. Else, let $b \in N(A') \setminus B'$ and a' one of its neighbours in A'.
3. If b is covered by M, say with a, then add the edges $a'b$ and ba to T. (See Figure 6.3). Then A' becomes $A' \cup \{a\}$ and B', $B' \cup \{b\}$. Go to Step 2. Else P, the (a_0, b)-path in T, is an M-augmenting path. Replace M by $M \triangle E(P)$. Go to Step 1.

6.2 Matching and vertex cover

We now show a duality theorem for the maximum matching in bipartite graphs.

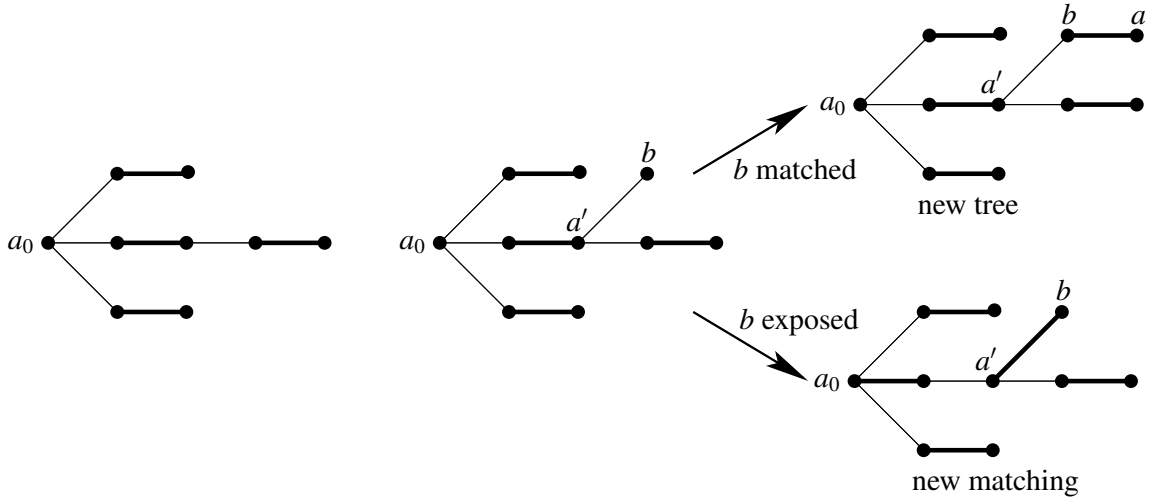


Figure 6.3: Step 3 of the Hungarian Method

Definition 6.5. Let $G = (V,E)$ be a graph. A set $K \subset V$ is a *vertex cover* of E if any edge of G is incident to a vertex in K. The *vertex cover number* of G, denoted $\nu(G)$, is the minimum size of a vertex cover of G.

Let K be a vertex cover of a graph. Then, for any matching M, K contains at least one endvertex of each edge of M. Hence, $|M| \leq |K|$. So, the maximum size of a matching is at most the minimum size of a vertex cover. For a bipartite graph, they are equal.

Theorem 6.6 (König 1931, Egerváry 1931)**.** *Let $G = ((A,B),E)$ be a bipartite graph. The size of a maximum matching equals the size of a minimum vertex cover, that is*

$$\mu(G) = \nu(G).$$

Proof. Let M be a maximum matching, let U be the set of exposed vertices in A, and let V' be the set of vertices of G linked to U using M-alternating paths. Let $A' = A \cap V'$ and $B' = B \cap V'$. See Figure 6.4. The set B' is saturated by M. Indeed, if a vertex $b \in B'$ is not matched, then the M-alternating path that links b to a vertex in U is an M-augmenting path, contradicting the maximality of M (Theorem 6.1). Moreover, $N(A') = B'$ by definition of V'. Let $K = B' \cup (A \setminus A')$ Then, any edge of G has an endvertex in K. Therefore, K is a vertex cover of G. But $|K| = |M|$ because $A \setminus A'$ is the set of vertices in A that are matched with some vertices in $B \setminus B'$. □

Hall's Theorem (6.2) can be deduced from this theorem.

Proof of Hall's Theorem (6.2): If G has no matching saturating A, then by Theorem 6.6, there is a vertex cover K with less than $|A|$ vertices. Let $A' = A \cap K$ and $B' = B \cap K$. Then, $|A'| + |B'| = |K| < |A|$ and

$$|B'| < |A| - |A'| = |A \setminus A'|$$

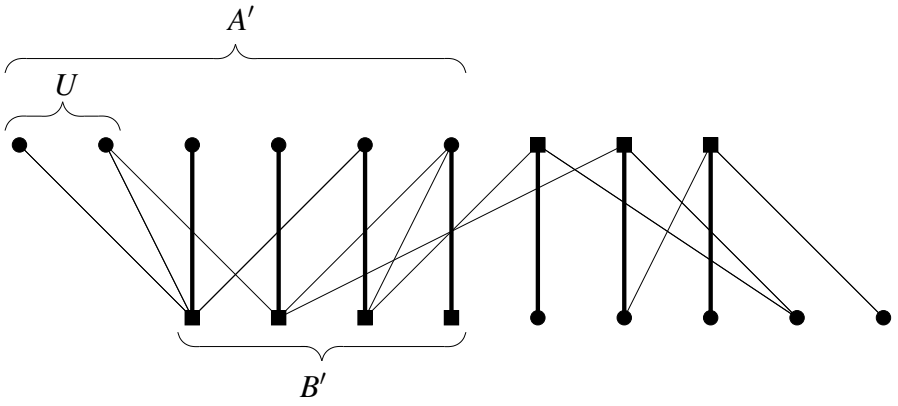


Figure 6.4: Finding a minimum vertex cover (squares) from a maximum matching (bold edges).

By definition of a vertex cover, there are no edges between $A \setminus A'$ and $B \setminus B'$, hence

$$|N(A \setminus A')| \leq |B'| < |A \setminus A'|$$

The set $A \setminus A'$ does not satisfies Hall's Condition (6.1). □

An *edge-cover* of a graph G is a set of edges $F \subset E(G)$ such that every vertex $v \in V(G)$ is incident to an edge $e \in F$. Note that an edge-cover can exist only if G has no isolated vertices. The *edge-cover number* of G, denoted $\rho(G)$, is the minimum size of an edge-cover of G.

Theorem 6.7 (Gallai, 1959)**.** *Let $G = (V,E)$ be a graph without isolated vertices. Then*

$$\alpha(G) + \nu(G) = |V| = \rho(G) + \mu(G).$$

Proof. By definition, S is a stable set if and only if $V \setminus S$ is a vertex cover. Hence $\alpha(G) + \nu(G) = |V|$.

Let M be a matching of size $\mu(G)$. For each of the $|V| - 2|M|$ vertices v missed by M, add to M and edge incident to v. We obtain an edge-cover of size $|M| + (|V| - 2|M|) = |V| - |M|$. Hence $\rho(G) \leq |V| - \mu(G)$. Let F be an edge-cover of size $\rho(G)$. For each $v \in V$ delete from F, $d_F(v) - 1$ edges incident to v. We obtain a matching of size at least $|F| - \sum_{v \in V}(d_F(v) - 1) = |F| - (2|F| - |V|) = |V| - |F|$. Hence $\mu(G) \geq |V| - \rho(G)$. □

Gallai's Theorem (6.7) and König-Egerváry Theorem (6.6) immediately imply the following.

Corollary 6.8 (König)**.** *Let G be a bipartite graph without isolated vertices. Then $\alpha(G) = \rho(G)$.*

6.3 Maximum-weight matching

Previous results on maximum matching in bipartite graphs can be generalized to maximum-weight matching in bipartite edge-weighted graphs. Note that, we may assume weights are non-negative.

The notion of vertex cover can be generalized to edge-weighted graphs

Definition 6.9. Let G be an edge-weighted graph. A *fractional vertex cover* is a function $c : V(G) \to \mathbb{R}^+$ such that, for any edge xy of G, $c(x)+c(y) \geq w(xy)$. The *weight* of a fractional vertex cover c of G is $c(G) = \sum_{v \in V(G)} c(v)$.

Let M be a matching and c be a fractional vertex cover in an edge-weighted graph (G,w). Then, the weight of c is at least the weight of M. Indeed, by summing all inequalities $c(x) + c(y) \geq w(xy)$ over all edges of M, we get $c(G) \geq w(M)$.

König-Egerváry Theorem is generalised to the weighted case:

Theorem 6.10. *In bipartite edge-weighted graph, the minimum weight of a fractional vertex cover equals the maximum weight of a matching.*

Proof. Let (G,w) be a bipartite edge-weighted graph. Free to add edges of weight 0, we may assume that $G = K_{n,n}$.

We provide an algorithm to finding a matching M and a fractional vertex cover c with same weight. Note that, for any edge xy of M, we have $c(x)+c(y) = w(xy)$.

Definition 6.11. Let c be a fractional vertex cover of (G,w). The *excess* of an edge xy is $c(x)+c(y)-w(xy)$. The *equality graph* of c, denoted by G_c, is the graph induced by the edges of excess 0.

Algorithm 6.3.

0. Initialization: Take the fractional vertex cover c defined by $c(a) := \max\{w(ab), ab \in E\}$ if $a \in A$ and $c(b) := 0$ if $b \in B$.
1. Find a maximum matching M in G_c.
2. If M is perfect in G, return "the maximum-weight matching is" M "and the minimum-weight fractional vertex cover is" c.
3. Else, let K be a fractional vertex cover with size $|M|$ in G_c. Let $R = A \cap K$ and $T = B \cap K$. Let $\varepsilon = \min\{c(a)+c(b)-w(ab) \mid a \in A \setminus R, b \in B \setminus T\}$. For any $a \in A \setminus R$, $c(a) := c(a) - \varepsilon$ and for any $b \in B \setminus T$, $c(b) := c(b) + \varepsilon$. Go to 1.

□

6.4 Matching in general graphs

Given a graph G, let $odd(G)$ denote the number of odd (i.e. with an odd number of vertices) connected components of G. Clearly, if G admits a perfect matching, then

$$odd(G-S) \leq |S| \qquad \text{for all} S \subset V(G).$$

Indeed, if G has a perfect matching, each odd component C of $G-S$ contains at least one vertex matched with a vertex not in C. Moreoever, this vertex must belong to S since there are no edges between distinct components of $G \setminus S$.

This necessary condition is actually sufficient.

Theorem 6.12 (Tutte 1947)**.** *A graph G admits a perfect matching if and only if $imp(G-S) \leq |S|$ for any $S \subset V(G)$.*

Proof. Let $G=(V,E)$ be a graph with no perfect matching. We will show that there is $S \subset V(G)$ such that $odd(G-S) > |S|$.

Let G' be the supergraph G with no perfect matching which has the maximum number of edges. For any S, a component of $G'-S$ is the union of components of $G-S$. Hence, an odd component of $G'-S$ contains an odd component of $G-S$. Hence, it is sufficient to find a *bad* set S, i.e., for which $odd(G'-S) > |S|$.

If G' contains a bad set, clearly it satisfies the Property $(\star)$.

Property $(\star)$: *S is complete, every component of $G'-S$ is complete, and every vertex of S is adjacent to all vertices of $G'-S$.*

Conversely, if a set satisfies Property $(\star)$ then S or $\emptyset$ is bad. Indeed, if $\emptyset$ is not bad, then $|V(G)|$ is even. If, in addition, S is not bad, then it is possible to matched (independently) a vertex of each odd component with a vertex in S and complete the matching in a perfect matching of the graph.

A vertex is *universal* if it is adjacent to every vertex but itself. Let S be the set of universal vertices in G'. We shall prove that S satisfies $(\star)$ and, so S is bad.

For sake of contradiction, assume that S does not satisfy $(\star)$. Then, a component of $G-S$ contains two non-adjacent vertices a and a'. Let a, b and c be the first three vertices in a shortest (a,a')-path in this component. Then, $ab, bc \in E(G')$ and $ac \notin E(G')$. Since b is not in S, a vertex of G', say d, ins not adjacent to b. By maximality of G', $G' \cup ac$ contains a perfect matching M_1 and $G' \cup bd$ has a perfect matching M_2. Note that $ac \in M_1$ and $bd \in M_2$ otherwise M_1 or M_2 would be a perfect matching in G'.

The graph induced by $M_1 \cup M_2$ is the union of even cycles alternating edges of M_1 and M_2. Let C be the cycle containing bd. If C does not contain the edge ac, then by replacing in M_2 the edges of $E(C) \cap M_2$ by the edges in $E(C) \cap M_1$, we get a perfect matching of G', a contradiction. If C contains ac, then, let P be the path in $C \setminus bd$ with start d and last edge ac. Without loss of generality, we may assume that a is the terminus of P. Let C' be the cycle obtained from $P \setminus a$ by adding the edges bc and bd. Replacing in M_2 the edges of $E(C') \cap M_2$ by the edges of $E(C') \cap M_2$, we get a perfect matching of G', a contradiction. □

6.5 Path-cover of digraphs

Consider a bipartite graph $G = ((A,B),E)$ and orient all its edges from A to B to get the digraph D. Then König-Egerváry's Theorem (Theorem 6.6) says how many disjoint directed paths are necessary to cover all vertices of D. Indeed, all directed paths have length 1 or 0, and the number of directed paths of such a "cover" is the smallest possible when it contains as many directed paths of length 1 as possible, i.e., a maximum matching.

In this section, we consider the following general question: given a digraph (non necessarily bipartite), how many paths are necessary to cover all its vertices?

Definition 6.13. A *path-cover* of a graph is a set of disjoint directed paths that cover all vertices. It can be viewed as a spanning forest every component of which is a directed path.

Recall that $\alpha(D)$ denotes the maximum size of a stable set in D.

Theorem 6.14 (Gallai and Milgram 1960)**.** *Every digraph D has a path-cover of at most $\alpha(D)$ directed paths.*

Proof. If P is a directed path, let us denote by $ter(P)$ its terminus. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two path-covers of D. We say that $\mathcal{P}_1 < \mathcal{P}_2$ if $\{ter(P) \mid P \in \mathcal{P}_1\} \subset \{ter(P) \mid P \in \mathcal{P}_2\}$ and $|\mathcal{P}_1| < |\mathcal{P}_2|$.

Let us show by induction that, if $\mathcal{P}$ is a path-cover which is minimum for $<$, then there is a stable set intersecting every directed path of $\mathcal{P}$.

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ be a path-cover minimum for $<$ in D. Let $v_i = ter(P_i)$ for any $1 \leq i \leq m$. If $\{v_i \mid 1 \leq i \leq m\}$ is a stable, then, the result holds. W.l.o.g., we may assume that (v_2, v_1) is an arc. Since the concatenation of P_2 and (v_2, v_1) is a directed path, by minimality of $\mathcal{P}$, v_1 is not the unique vertex of P_1. Let v be the vertex preceding v_1 in P_1 and set $P_1' = P_1 - v_1$. Then $\mathcal{P}' = \{P_1', P_2, \ldots, P_m\}$ is a path-cover of $D - v$.

Let us show that $\mathcal{P}'$ is minimum for $<$ in $D - v$. Assume that there is a path-cover $\mathcal{Q}' < \mathcal{P}'$. If $\mathcal{Q}$ contains a directed path P with terminus v or v_2, then replacing P by the concatenation of P and (v, v_1) or (v_2, v_1), we get a path-cover $\mathcal{Q} < \mathcal{P}$, a contradiction. If not, then $\mathcal{Q}'$ consists of at most $m-2$ directed paths and $\mathcal{Q}' \cup \{(v_1)\} < \mathcal{P}$, a contradiction.

Hence, $\mathcal{P}'$ is minimum for $<$ in $D - v$. So, by the induction hypothesis, $D - v$ admits a stable set intersecting every directed path of $\mathcal{P}'$. Clearly, this stable set also intersects every directed path of $\mathcal{P}$. □

Remark 6.15. In 1990, Hahn and Jackson [2] conjectured that this theorem is best possible in the following strong sense. *For each positive integer k, there is a digraph D with stability number k such that deleting the vertices of any $k-1$ directed paths in D leaves a digraph with stability number k.* This was recently proved by Fox and Sudakov [1].

Definition 6.16. An *ordered set* is a pair $(P, \leq)$ such that P is a set and $\leq$ is a binary relation over P that is *reflexive* (for all $x \in P$, $x \leq x$), *antisymmetric* (if $x \leq y$ and $y \leq x$ then $x = y$) and *transitive* (if $x \leq y$ and $y \leq z$ then $x \leq z$). Two elements x and y of P are *comparable* if $x \leq y$ or $y \leq x$. A *chain* of $(P, \leq)$ is a set of pairwise comparable elements, and an *antichain* of $(P, \leq)$ is a set of pairwise non-comparable elements.

Corollary 6.17 (Dilworth 1950)**.** *For any ordered set $(P,\leq)$, the minimal number of chains covering P equals the maximum size of an antichain.*

Proof. If A is an antichain of maximum size in $(P,\leq)$, then at least $|A|$ chains are necessary to cover P because any chain contains at most one element of A. To show that $|A|$ chains are sufficient, consider the digraph $D=(P,E_<)$ with $E_< = \{(x,y)|x \leq y \text{ and } x \neq y\}$. Clearly, there is a one-to-one correspondence between the chains of $(P,\leq)$ and the directed paths of D, and there is a one-to-one correspondence between the antichains of $(P,\leq)$ and the stable sets of D. Hence Theorem 6.14 applied to D yields the result. □

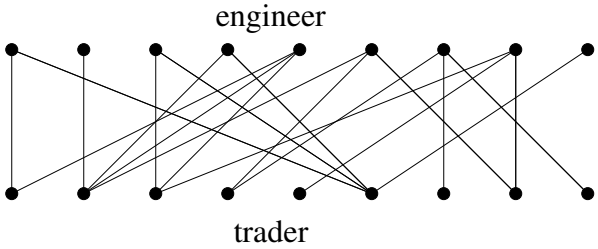
6.6 Exercises

Exercise 6.1. The french figure skating federation wants to form couples (one girl and one boy) for dance on ice in order to prepare the Olympic Winter Games. Six girls and six boys of high enough level volunteer. In view of temper incompatibility between some girls and some boys as well as aestethic criterias supposedly displeasing the judges (the federation does not have enough money to corrupt all judges), the following table has been designed. A cross in a square means that the two iceskaters cannot form a couple.

	Girl 1	Girl 2	Girl 3	Girl 4	Girl 5	Girl 6
Boy 1	×	×		×	×	
Boy 2						×
Boy 3	×			×	×	×
Boy 4	×				×	×
Boy 5		×	×			
Boy 6	×		×	×	×	

How many couples at most can the federation form? Justify your answer.

Exercise 6.2. A managing director has to launch the marketing of a new product. Several candidate products are at his disposal and he has to choose the best one. Hence, he let each of these products be analysed by a team made of an engineer and a trader who write a review together. The teams are made along the following graph; each edge corresponds to a product and its endvertices to the engineer and trader examining it.



How many people at least does the managing director gather in order to have the report on all the products? (The report can be given by either the engineer or the trader.)

Exercise 6.3.
1) Show that a tree has at most one perfect matching.
2) Show that a tree T has a perfect matching if and only if $odd(T-v)=1$ for all vertex v.

Exercise 6.4. Let $G=((A,B),E)$ be a bipartite graph.
1) Prove that the number of edges in a maximum matching is $|A|-\max_{A'\subset A}\{|A'|-|N(A')|\}$.
2) Deduce that if $|A|=|B|=n$ and $|E(G)|>(k-1)n$ then G has a matching of cardinality k.

Exercise 6.5. Let $G=((A,B),E)$ a bipartite graph such that $|N(S)|>|S|$ for all subset S of A distinct from $\emptyset$ and A. Show that every edge $e\in E(G)$ is in a matching saturating A.

Exercise 6.6. Let $G=((A,B),E)$ be a bipartite graph. Suppose that $S\subseteq A$, $T\subseteq B$ and G contains a matching M_1 saturating S and a matching M_2 saturating T. Prove that there exists a matching which saturates both S and T.

Exercise 6.7. Let $G=((A,B),E)$ be a bipartite graph such that $|A|=|B|=n$ and $\delta(G)\geq n/2$. Show that G has a perfect matching.

Exercise 6.8. Let $G=((A,B),E)$ be a bipartite graph and W be the set of vertices with minimum degree in G.
1) Prove that G contains a matching saturating $A\cap W$.
2) Deduce that there exists a matching saturating W. (*One could use Exercice 6.6*).

Exercise 6.9. A pack of $m\times n$ cards with m values and n colours is made of one card of each value and colour. The cards are arranged in an array with n lines and m columns. Show that there exists a set of m cards, one in each column, such that they all have distinct values.

Exercise 6.10. Prove that a bipartite graph G has a matching of size at least $|E(G)|/\Delta(G)$.

Exercise 6.11. Let $G=((A,B),E)$ be a connected bipartite graph such that $|A|=|B|=2k+3$. Show that if all the vertices have degree k or $k+1$ then G has a perfect matching unless it is a graph H which has an edge e such that $G\setminus e$ has two connected components which are subgraphs of $K_{k+2,k+1}$.

Exercise 6.12 (Alon)**.** Let $G=((A,B),E)$ be a bipartite graph such that $d(a)\geq 1$ for all $a\in A$ and $d(a)\geq d(b)$ for all $ab\in E$, where $a\in A$ and $b\in B$. Show that G has a matching saturating A.

Exercise 6.13 (Alon)**.** Let $G=((A,B),E)$ be a bipartite graph in which each vertex of A is of odd degree. Suppose that any two vertices of A have an even number of common neighbours. Show that G has a matching saturating A.

Exercise 6.14 (Petersen's Theorem)**.** Let G be a cubic graph.
1) Prove that if G is 2-edge-connected then it has a perfect matching.
2) Give an example of cubic graph with no perfect matching.
3)

a) Prove that if G is 2-edge-connected then, for all edge e, $G \setminus e$ has a perfect matching.

b) Deduce that if G has a unique bridge (separating edge) then G has a perfect matching.

Exercise 6.15. Let G be a connected graph.
1) Prove that if $|V(G)| \geq 4$ and every edge of G is in a perfect matching then G is 2-connected.
2) More generally, show that if $|V(G)| \geq 2k$ and every set of $k-1$ independent edges is included in a perfect matching then G is k-connected.

Exercise 6.16. Let G be a graph on at least $2k+2$ vertices which has a perfect matching. Show that if every set of k independent edges is included in a perfect matching then every set of $k-1$ independent edges is included in a perfect matching.

Exercise 6.17. Let D be a digraph. A *cycle-factor* of D is a spanning subdigraph F of D such that $d_F^+(v) = d_F^-(v) = 1$ for all $v \in V(D)$.

The *bipartite associated to D*, denoted $B(D)$, is defined by

- $V(B(D)) = V(G) \times \{1,2\}$ and
- $E(B(D)) = \{((u,1),(v,2)) \mid uv \in A(D)\}$

1) Show that D has a cycle-factor if and only if $B(D)$ has a perfect matching.
2) Assume that for every vertex v of D, $d^+(v) = d^-(v) = k$ for some fixed $k > 0$. Show that D has a cycle-factor.
3) Let G be a $2k$-regular graph. A *2-factor* in G is a spanning 2-regular graph. Deduce from the previous question that G has a 2-factor.

Exercise 6.18. Two persons are playing the following game on a graph. One after another the players choose vertices (one per turn) $v_1, v_2, v_3, \ldots$ so that v_i is adjacent to v_{i-1} for all $i \geq 0$. The last player which is able to choose a vertex wins.

Prove that the first player has a winning strategy if and only if the graph has no perfect matching.

Exercise 6.19. The *claw* is the graph $K_{1,3} = (\{x, y_1, y_2, y_3\}, \{xy_1, xy_2, xy_3\})$. Show that an even connected graph with no claw as an induced subgraph has a perfect matching.

Exercise 6.20. Let G be a connected graph with an even number of edges. Prove that $E(G)$ may be partitionned into $|E(G)|/2$ sets of two adjacent edges. *One could show that a "line graph" with an even number of vertices has a perfect matching.*

Exercise 6.21 (Erdős ans Szekeres)**.** Prove that a sequence of $rs+1$ integers contains an increasing subsequence of $r+1$ integers or a decreasing subsequence of $s+1$ integers.

Exercise 6.22. Let G be a graph with vertices $v_1, \ldots, v_n$. Give an algorithm that, given a sequence $d_1, \ldots, d_n$, decides in polynomial time if G admits an orientation such that $d^+(v_i) = d_i$ for all $1 \leq i \leq n$.

Bibliography

[1] J. Fox and B. Sudakov. Paths and stability number in digraphs. *manuscript*.

[2] G. Hahn and B. Jackson. A note concerning paths and independence number in digraphs. *Discrete Math.* 82:327–329, 1990.

Part II

Lecture Notes (1st Term)

Chapter 7

Flow Problems

7.1 Introduction and definitions

Problems related to transport have been investigated since the early fifties. The problem is to route some goods, called *commodities*, from production sites to consumption sites, through a network consisting of communication links inter-connecting the sites (pipe-lines, routes, telecommunication networks). Moreover, each link has a maximum admissible throughput, called the *capacity* of the link.

In general, most of the sites of the network do not produce nor consume anything, they are only used to interconnect links. To each production or consumption site is associated some real number that corresponds to the maximum production or consumption.

The main objective is to maximize the throughput of the traffic.

We first consider networks with directed links, i.e., that can be used in one direction. Formally, a *flow network* is defined as follows.

Definition 7.1 (Flow network)**.** A *flow network* is a four-tuple $(G, pr_{max}, co_{max}, c)$ such that:

- G is a digraph (or a mulitdigraph), the vertices of which represent the sites and the arcs represent the links;
- pr_{max} is a function from $V(G)$ to $\mathbb{R}^+ \cup +\infty$; $pr_{max}(v)$ corresponds to the maximum production possible in v. If v is not a production site, then $pr_{max}(v) = 0$.
- co_{max} is a function from $V(G)$ to $\mathbb{R}^+ \cup +\infty$; $co_{max}(v)$ corresponds to the maximum consumption possible in v. If v is not a consumption site, then $co_{max}(v) = 0$.
- c is a function from $E(G)$ to $\mathbb{R}^+ \cup +\infty$; $c(e)$ corresponds to the capacity of the link e.

Definition 7.2 (Flow)**.** The *flow* is a triple $F = (pr, co, f)$ where

- pr is a function of production such that, for any vertex v, $0 \leq pr(v) \leq pr_{max}(v)$;

- co is a function of consumption such that, for any vertex v, $0 \leq co(v) \leq co_{max}(v)$;

- f is a function over E, called *flow function* that satisfies the following constraints:

Positivity:	$\forall e \in E(G)$,	$f(e) \geq 0$
Capacity constraint:	$\forall e \in E(G)$,	$f(e) \leq c(e)$
Flow conservation:	$\forall v \in V(G)$,	$\sum_{(u,v)\in E(G)} f((u,v)) + pr(v) = \sum_{(v,u)\in E(G)} f((v,u)) + co(v)$

By summing, the $|V(G)|$ equations of flow conservation, we get:

$$\sum_{v\in V(G)} pr(v) = \sum_{v\in V(G)} co(v)$$

In other words, the sum of all produced commodities equals the sum of all consumed commodities. This value corresponds to the amount of routed traffic, it is the *flow value*, denoted by $v(F)$.

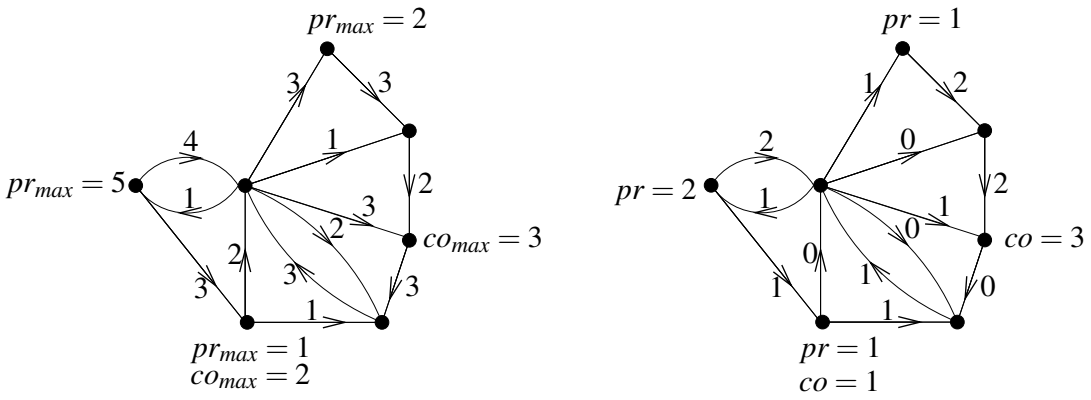


Figure 7.1: Example of flow network (left) and a flow of value 4 in it (right)

Hence, the problem is the following:

Problem 7.3 (Maximum Flow)**.**
Instance: a flow network N.
Find: a flow with maximum value.

A flow with maximum value is said to be a *maximum flow*.

7.2 Reducing to an elementary network

Before studying this problem, we show that it is equivalent to consider an elementary problem where there is a unique production site with infinite maximum production and a unique consumption site with infinite maximum consumption.

Definition 7.4 (Elementary network)**.** A flow network $(G, \infty_s, \infty_t, c)$ is *elementary*:

- s and t are two distinct vertices where s is a *source* (i. e. $d^-(s) = 0$) and t is a *sink* (i.e. $d^+(t) = 0$);
- $\infty_s(s) = +\infty$ and $\infty_s(v) = 0$ for all $v \neq s$;
- $\infty_p(t) = +\infty$ and $\infty_p(v) = 0$ for all $v \neq t$.

We denote $(G, \infty_s, \infty_t, c)$ by (G, s, t, c).

Definition 7.5 (Associated elementary network)**.** If $N = (G, pr_{max}, co_{max}, c)$ is a flow network, its *associated* elementary network is the following elementary network $N = (\bar{G}, s, p, \bar{c})$ obtained from N in the following way:

- add a source s and a sink t;
- link s to all vertices v with non-null production with an arc (s, v) of capacity $pr_{max}(v)$;
- link all vertices v with non-null consumption to t with a link (v,t) of capacity $co_{max}(v)$;

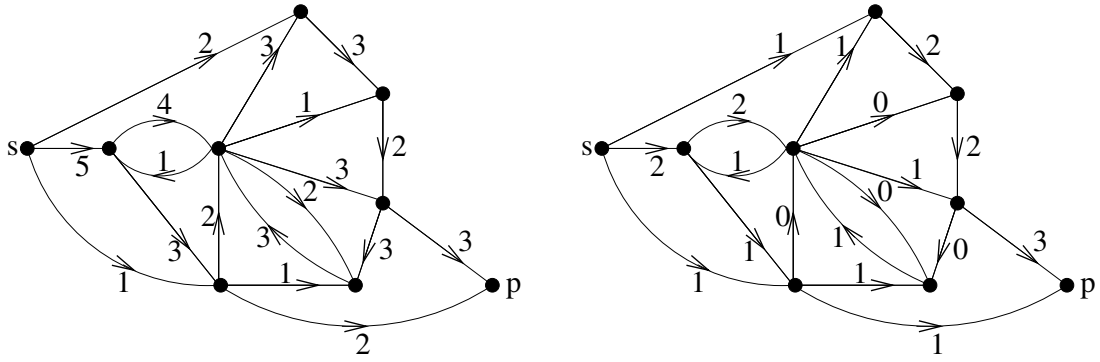


Figure 7.2: Elementary network associated to the flow network depicted in Figure 7.1 (right) and the flow in this network corresponding to the one of Figure 7.1 (left)

It is easy to see that there is a one-to-one correspondence between any flow $F = (pr, co, f)$ of the elementary network and a flow $F' = (pr', co', f')$ of the initial problem defined as follows: for any production site u, $pr'(u) = f((s,u))$; for any consumption site v, $co'(v) = f((v,t))$ and for any arc (x,y), $f'((x,y)) = f((x,y))$.

Clearly, both flows have same value $v(F) = v(F') = pr(s) = co(t)$. Hence:
Finding a maximum flow in a flow network is equivalent to finding a maximum flow solve in its associated elementary network.

In the following, we only consider elementary flow networks.

Note that, in an (elementary) flow network $N = (G, s, t, c)$, the flow conservation constraint can be written:

$$\forall v \in V(G) \setminus \{s,t\}, \sum_{(u,v)\in E(G)} f((u,v)) = \sum_{(v,u)\in E(G)} f((v,u))$$

Moreover, a flow $F = (pr, co, f)$ of an (elementary) flow network is well defined by the flow function f since, by the flow conservation constraint in s and t, we have

$$\begin{aligned} v(F) &= pr(s) = \sum_{u\in V(G),(s,u)\in E(G)} f((s,u)) \\ &= co(t) = \sum_{u\in V(G),(u,t)\in E(G)} f((u,t)) \end{aligned}$$

Therefore, for ease of presentation, we often identify a flow F with its function f, and we note the flow value by $v(f)$.

To simplify the notations in the sequel, for a flow f or a capacity c, and an arc (u,v), we write $f(u,v)$ instead of $f((u,v))$, and $c(u,v)$ instead of $c((u,v))$.

7.3 Cut and upper bound on the maximum flow value

We will show that the value of a maximum flow in a network is limited by the existence of some bottlenecks through which the traffic must go. Roughly, to go from the source to the sink, the flow must cross the border of a set of vertices, its size (the sum of the capacities of the corresponding edges) will limit the value of the flow. Such a border is called a *cut*.

Definition 7.6 (Cut)**.** In a flow network $N = (G, s, t, c)$, an (s,t)-*cut*, or simply a *cut*, is a bipartition $C = (V_s, V_t)$ of the vertices of G such that $s \in V_s$ and $t \in V_t$. The arcs form V_s to V_t (i.e. with tail in V_s and head in V_t) are the *arcs of* C. Their set is denoted by $E(C)$. The *capacity* of the cut C, denoted by $\delta(C)$, is the sum of the capacities of its arcs: $\sum_{e\in E(C)} c(e)$.

Let f be a flow and $C = (V_s, V_t)$ be a cut, $out(f,C)$ denotes the flow on arcs leaving V_s and $in(f,C)$ the flow entering V_s :

$$\begin{aligned} out(f,C) &= \sum_{(u,v)\in E(C),\ u\in V_s, v\in V_t} f(u,v) \\ in(f,C) &= \sum_{(u,v)\in E(G),\ u\in V_t, v\in V_s} f(u,v) \end{aligned}$$

Note that the flow conservation implies that

$$\text{for all cut } C, \quad v(f) = out(f,C) - in(f,C)$$

In particular, the value of the flow is always at most $out(f,C)$. But clearly $out(f,C) \leq \delta(C)$,so

$$v(f) \leq \delta(C). \tag{7.1}$$

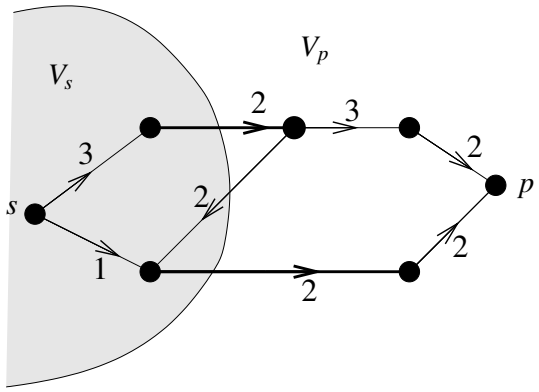


Figure 7.3: A flow network and a cut with capacity 4. Bold arcs are those of the cut

A cut can be viewed as a set of arcs that the flow must cross and whose capacity limits the value of the flow.

Let $v_{max} = max\{v(f), f \text{ a flow}\}$ and $\delta_{min} = min\{\delta(C),\ C \text{ a cut}\}$. Equation 7.1 applied to a maximum flow and a minimum-capacity cut yields

$$v_{max} \leq \delta_{min}. \tag{7.2}$$

In fact, the next theorem,due to Ford and Fulkerson, states that there is equality.

Theorem 7.7 (FORD–FULKERSON THEOREM)**.** *The maximum value of an (s,t)-flow equals the minimum capacity of an (s,t)-cut, i.e.,*

$$v_{max} = \delta_{min}.$$

We often say that *max flow equals min cut*. It is an example of"min-max theorem"

There are many proofs of this theorem, some being non-constructive proof of this theorem. In the next section, we present the original proof which is based on an algorithm computing a maximum flow and a cut with minimum capacity.

7.4 Auxiliary Network and "push" Algorithm

Most of the algorithms for computing maximum flows are based on the following idea: Starting from an existing flow (initially, it may be null), the flow is increased by going from the source to the sink by "pushing" the commodity where it is possible.

The difference between the algorithms mainly consists of the way used to decide where and how to push some flow.

For this purpose, we define an *auxiliary network*. This graph, denoted by $N(f)$, depends on the existing flow f.

Definition 7.8 (Auxiliary network)**.** Given a flow network $N = (G,s,t,c)$ and a flow f, we build the auxiliary network $N(f) = (G(f),s,t,c_f)$ as follows.
For any pair of vertices (u,v), let

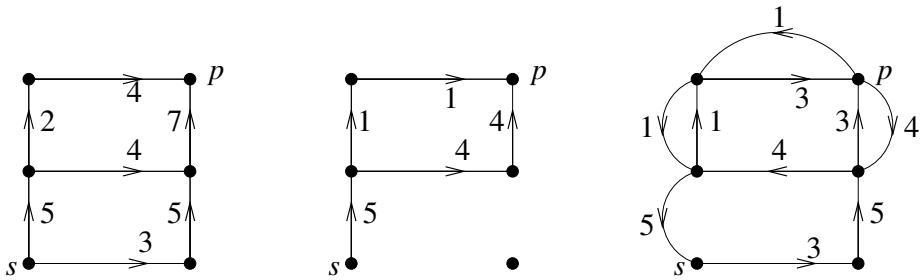
$$c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$$

with $c(u,v)$, $f(u,v)$ and $f(v,u)$ equal to 0 when they are not defined (if (u,v) is not an arc of G). Note that $c_f(u,v) \geq 0$ since $c(u,v) - f(u,v) \geq 0$. Then, $G(f)$ is defined as follows

$$\begin{aligned} V(G(f)) &= V(G) \\ E(G(f)) &= \{(u,v) \mid c_f(u,v) > 0\} \end{aligned}$$

Note that $c_f(u,v) + c_f(v,u) = c(u,v) + c(v,u)$. Intuitively, $c_f(u,v)$ is the sum of the remaining capacity on the arc (u,v), i.e., $c(u,v) - f(u,v)$, plus a virtual capacity $f(v,u)$, that allows to "remove" some flow on the arc (v,u), which corresponds to virtually push some flow along (u,v). See examples in Figures 7.4 and 7.5.

Note that the auxiliary network has no arc with capacity 0. This is important because it ensures that, for any directed path P in $G(f)$, the minimum capacity of the arcs of P is positive.



Flow network Flow Auxiliary network

Figure 7.4: A flow network, a flow and the corresponding auxiliary network

Let P be a directed (s,t)-path in $N(f)$ where the minimum capacity of an arc in P is $\varepsilon > 0$. The flow f' obtained by *pushing* ε *units of flow* along P is defined by:
For any arc $(u,v) \in E(P)$, we push the flow along (u,v), that is, we increase the flow with ε units on (u,v). Two cases may happen:

- If the remaining capacity of (u,v) is sufficient for the ε units of flow, i.e., $f(u,v) + \varepsilon \leq c(u,v)$, then $f'(u,v) = f(u,v) + \varepsilon$ and $f'(v,u) = f(v,u)$.

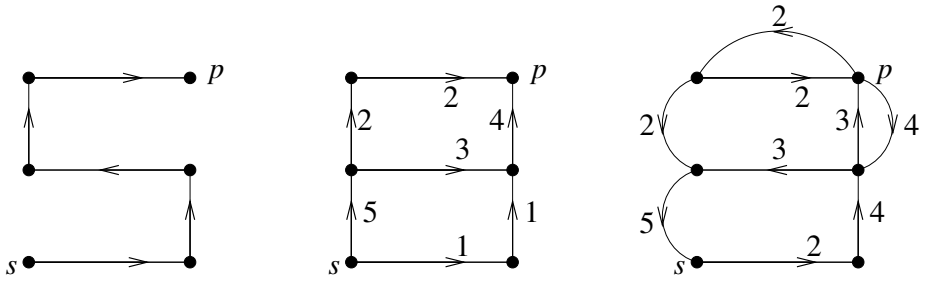
- If the remaining capacity of (u,v) is not sufficient for the ε units of flow, i.e., $f(u,v) + \varepsilon > c(u,v)$, then, by definition of the auxiliary network, we have $f(v,u) \geq f(u,v) + \varepsilon - c(u,v) > 0$. Hence, (v,u) has a flow excess of $f(u,v) + \varepsilon - c(u,v)$ units that we must remove. We set $f'(u,v) = c(u,v)$ and $f'(v,u) = f(v,u) - (f(u,v) + \varepsilon) + c(u,v)$.

If (u,v) and (v,u) do not belong to P, the flow remains unchanged on these arcs, $f'(u,v) = f(u,v)$ and $f'(v,u) = f(v,u)$.

Lemma 7.9. *If f is a flow of value $v(f)$, then f' is a flow of value $v(f)+\varepsilon$.*

Proof. See Exercice 7.5. □

For instance, from the flow depicted in Figure 7.4, by taking the directed (s,t)-path depicted in Figure 7.5 with minimum capacity 1 and pushing the flow, we obtain the new flow and the new auxiliary network depicted in Figure 7.5.



(s,p)-path New flow New auxiliary network

Figure 7.5: Push along a directed (s,t)-path of the auxiliary network in Figure 7.4.

Now, we have some elements of an algorithm:

1. Start with any flow f;
2. Compute the auxiliary network $N(f)$;
3. Find a directed path from s to p in $G(f)$;
4. If such a directed path P exists, then push some flow along P and update f.

Note that, if there is no directed path from the source to the sink in the auxiliary graph, the previous algorithm does nothing. We will see that, in this case, the existing flow is maximum.

Proposition 7.10. *If $G(f)$ does not contain a path from the source to the sink, then the flow f is maximum.*

Proof. Let V_s be the set of all vertices that can be reached from s in $G(f)$. V_s does not contain the sink since there are no (s,t)-paths. Hence, $p \in V_t = V \setminus V_s$. Let C be the cut (V_s, V_t).

Let (u,v) be an arc of C. By definition of V_s, the arc (u,v) is not in $G(f)$. So, by definition of $G(f)$, this means that $c_f(u,v) = 0$. Hence, f is such that $f(u,v) = c(u,v)$ and $f(v,u) = 0$.

We get that $out(f,C) = \delta(C)$ and $in(f,C) = 0$. Informally, there is no flow entering V_s and all arcs of C (that is leaving V_s) are saturated.

But $v(f) = out(f,C) - in(f,C)$, therefore

$$v(f) = \delta(C)$$

The current flow has the same value as the capacity of the cut C. Hence, f is maximum by Equation (7.2). □

Note that this proof also exhibit a cut (V_s, V_t) with same capacity as the maximum value of a flow. Hence it is a minimum-capacity cut. It shows that it is easy to find a minimum-capacity cut (V_s, V_t) from a maximum flow: V_s is the set of the vertices reachable from the source in the auxiliary network and V_t its complement.

Now, we can prove Theorem 7.7.

Proof of Theorem 7.7: Let f be a flow with maximum value in a flow network (G,s,t,c), and let $N(f)$ be the auxiliary network. In $G(f)$, there are no directed (s,t)-paths, otherwise we obtain a flow with greater value by pushing some flow along this path (Lemma 7.9). Hence, by the proof of Proposition 7.10, the cut $C = (V_s, V_t)$, where V_s is the set of vertices v such that there is a directed (s,v)-path in $G(f)$, has capacity $v(f) = v_{max}$. Therefore, $\delta_{min} \leq \delta(C) \leq v_{max}$. □

7.5 Ford-Fulkerson algorithm

We now have the following algorithm:

Algorithm 7.1 (Ford and Fulkerson (1956))**.**

1. Start with null flow $f = 0$;
2. Compute the auxiliary network $N(f)$;
3. Look for a directed path from s to t in $G(f)$;
4. If such a directed path P exists, then push some flow along P, update f and go to 2;
5. Else terminate and return f.

This algorithm is correct in the sense that If the algorithms terminates, then it returns a maximum solution. But

1) does it always terminate?

2) If it terminates, what is its complexity?

The answer to question 1 is somehow yes and no: we will see that the algorithm always terminates if the capacities are rational, but it can take infinite time if capacities are real. Besides, even when the capacities are integers, the running time depends linearly on the value of the maximum flow that may be huge.

Analysis of Ford-Fulkerson Algorithm

Proposition 7.11. *(i) If all capacities are integers, then Algorithm 7.1 terminates after at most v_{max} searches of a directed path.*

(ii) If all the capacities are rational, then Algorithm 7.1 terminates after at most $\mu \cdot v_{max}$ searches of a directed path, with μ the least common multiple of the denominators of the capacities.

Proof. (i) If capacities are integers, at each iteration, the algorithm pushes at least one unit of flow (since, in this case, ε is integral). Each iteration requires the search of a path from s to p in $G(f)$. This can be done in time $O(|E(G)|)$ by any search algorithm (See Chapter 2). Hence, its total running time is at most

$$O(\, v_{max} \cdot |E(G)|).$$

(ii) If capacities are rational, the algorithm terminates since the flow increases of at least $\frac{1}{\mu}$ at each iteration. μ can be very large, but it is fixed. Actually, we can solve the problem by multiplying all capacities by μ and by solving the integral problem obtained: it is proportional the initial one.

□

Remark 7.12. If some capacities are integers, then the Ford-Fulkerson algorithm returns an integral maximum flow.

Proposition 7.13. *If the capacities are real, the algorithm may not terminate. Moreover, the increasing sequence of the flow value may converge to a value a lot smaller than the optimal value.*

Proof. Consider the flow network shown in Figure 7.6 for which the capacities of all edges are infinite (or if the reader prefers, a huge integer M). Let α denote the positive root of $x^3 + x - 1 = 0$. Clearly $1/2 < \alpha < 1$. Let the initial flow f_0 be as shown in Figure 7.6.

We shall prove that by employing a well (or very badly) chosen sequence of four augmenting paths over and over, Algorithm 7.1 will produce an infinite sequence of flows, the values of which are monotone increasing and which converge to a limit not exceeding 16. For any $m \geq 0$, start from the flow f_{4m} and push along the directed path (s,c,d,a,b,t). The pushed amount is α^{4m+1} because of arc (a,b) in the auxiliary network. The resulting flow is f_{4m+1}. Push along the directed path (s,c,b,a,d,t) an amount of α^{4m+2} (because of (c,b)) to produce f_{4m+2}, push along the directed path (s,a,b,c,d,t) an amount of α^{4m+3} (because of (c,d)) to produce

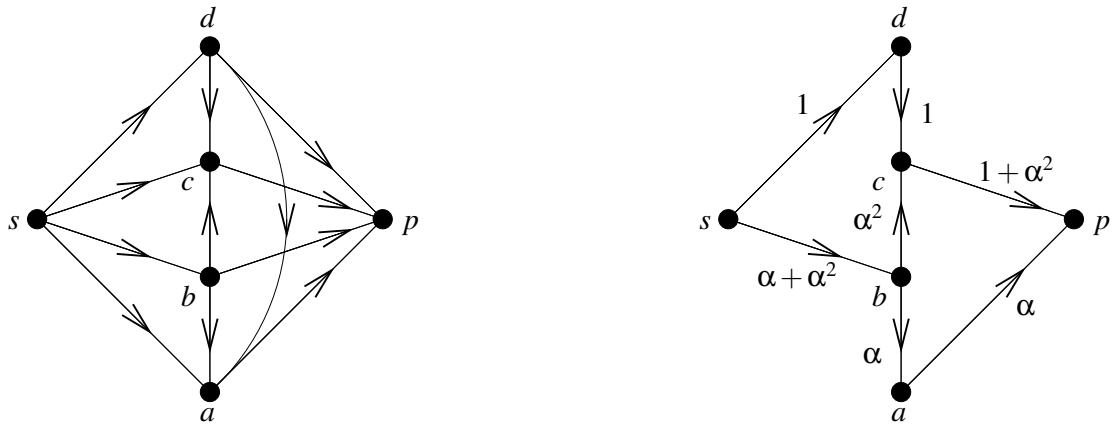


Figure 7.6: A flow network and the initial flow f_0

f_{4m+3} and push along the directed path (s,a,d,c,b,t) an amount of α^{4m+4} (because of (a,d)) to produce f_{4m+4}. See Figure 7.7.

For $k \geq 1$, the augmentation of the value from f_{k-1} to f_k is α^k and hence

$$\begin{aligned}
v(f_k) &= v(f_0) + \alpha + \alpha^2 + \cdots + \alpha^r \\
&= (1 + \alpha + \alpha^2) + \alpha + \alpha^2 + \cdots + \alpha^r \\
&= \frac{1}{\alpha}(\alpha + \alpha^2 + \alpha^3 + \alpha^2 + \alpha^3 + \cdots + \alpha^{r+1}) \\
&= \frac{1}{\alpha}(\alpha + \alpha^2 + (1-\alpha) + \alpha^2 + \alpha^3 + \cdots + \alpha^{r+1}) \\
&= \frac{1}{\alpha}(1 + 2\alpha^2 + \alpha^3 + \cdots + \alpha^{r+1}) \\
&< \frac{1}{\alpha}(1 + \alpha + \alpha^2 + \alpha^3 + \cdots + \alpha^{r+1}) \\
&< \frac{1}{\alpha - \alpha^2} = \frac{1}{\alpha^4} < 16.
\end{aligned}$$

From this construction, it is easy to construct an example of a network where some capacities are irrational such that even starting with a null flow, the sequence of flows obtained by pushing along some directed paths converges to a value less than 16, while the value of a maximum flow is infinite (or arbitrarily large). See Exercise 7.9.

□

Remark 7.14. While the termination is not ensured in case of irrational capacities, Theorem 7.7 ($v_{max} = \delta_{min}$) remains valid. We prove the real case by taking the limit of the rational case. In practice, the rational case is the only important problem, since computers works with a finite precision.

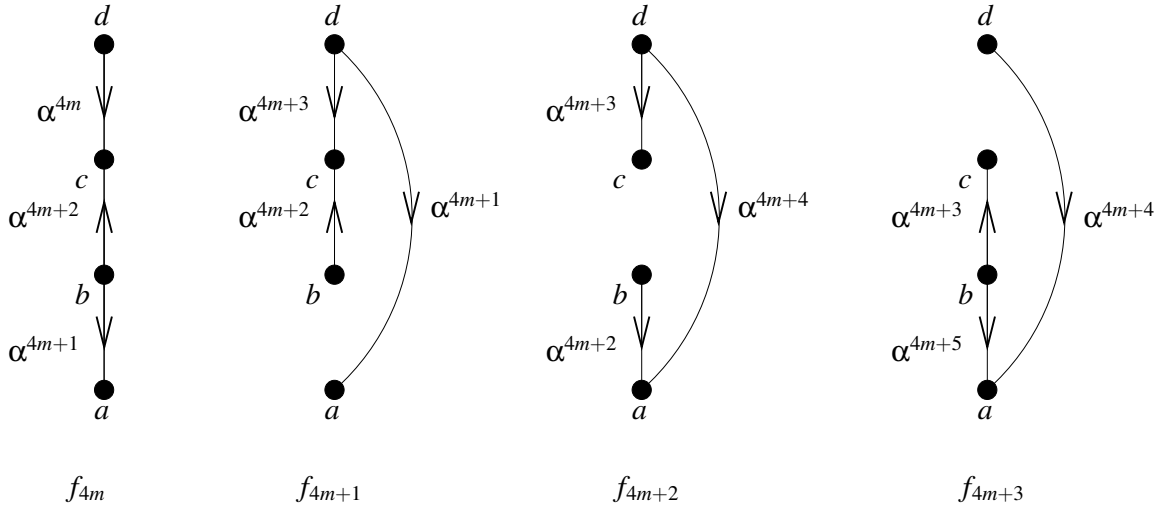


Figure 7.7: The sequence of flows. The arcs leaving s and the arcs entering t are not drawn

The bound of Proposition 7.11 on the number of pushes is not good because it depends on the value of the maximum flow which may be huge. Figure 7.8 shows an example where v_{max} pushes, if they are badly chosen, are necessary to reach a flow with maximum value. Indeed, if

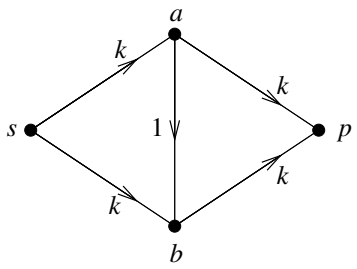


Figure 7.8: Example where $2k = v_{max}$ pushes may be performed.

the pushes are alternatively performed along $P_1 = (s,a,b,t)$ and $P_2 = (s,b,a,t)$, then one unit of flow is pushed at each iteration because the capacity of (a,b) or (b,a) equals 1. Therefore, $2k = v_{max}$ pushes are necessary (k along P_1 and k along P_2).

To improve the previous algorithm and to ensure its quick termination in all cases, the idea is to consider the pushes on some specific directed paths.

7.6 Pushing along shortest paths

Instead of pushing the flow along an arbitrary directed path (s,t)-path in the auxiliary network, a better algorithm consists in pushing along a shortest directed (s,t)-path. Here, by shortest

path, we mean a shortest path in terms of number of arcs, not of total capacity.

Algorithm 7.2 (Edmonds-Karp, 1970)**.**

1. Start with null flow $f = 0$;
2. Compute the auxiliary network $N(f)$;
3. Look for a shortest directed path from s to p in $G(f)$;
4. If such a path P exists, push some flow along P, update f and go to 2;
5. Else terminate and return f.

We will show that such an algorithm performs at most $\frac{|E(G)||V(G)|}{2}$ iterations. That is, its time complexity is at most $\frac{|E(G)||V(G)|}{2}$ times the complexity of finding a shortest path. Note that, contrary to the Ford-Fulkerson Algorithm (7.1), this bound is independent from the capacities.

The proof is based on two simple properties.

1) During the iterations, the distance between s and t in the auxiliary graph cannot decrease.

2) During the iterations, the distance between s and t remains unchanged at most $|E(G)|$ consecutive iterations.

Let f_0 be a flow and let $G_0 = G(f_0)$; We set:

- E' the set of arcs of G_0 that belong to a shortest path from s to t in G_0;
- E'^{-} the arcs obtained by reversing the ones in E' ;
- G_1 the graph obtained from G_0 by adding the arcs in E'^{-}.

Lemma 7.15. *The graph G_1 has the following properties:*

(i) A directed path from s to t with length $dist_{G_0}(s,t)$ does not use any arc in E'^{-}.

(ii) The distance from s to p in G_1 is $dist_{G_0}(s,t)$;

Proof. (i) Let P be a directed (s,t)-path in G_1 that contains some arcs in E'^{-}, and let (v,u) be the last arc of E'^{-} that belongs to P. Let us show that P is not a shortest directed (s,t)-path. Since (u,v) belongs to a shortest directed (s,t)-path, we have $dist_{G_0}(u,p) = dist_{G_0}(v,p) + 1$. Since between u and p, the path P uses only arcs in G_0, its length is $l + 1 + dist_{G_0}(u,p)$ (where l is the length of the subpath of P between s and v). Hence, P has length $l + 2 + dist_{G_0}(v,p)$. The path P' obtained by following P until v and then using a shortest directed (v,p)-path has length $l + dist_{G_0}(v,p) = l + dist_{G_0}(v,p)$. Therefore, P is not a shortest directed (s,t)-path.

(ii) Follows from (i). □

Theorem 7.16. *Algorithm 7.2 performs at most $|E(G)||V(G)|$ iterations. Its time-complexity is $|E(G)||V(G)|$ searches of shortest directed paths, so $O(|E(G)|^2|V(G)|)$.*

Proof. During a push, we add to the auxiliary network the arcs in E'^{-}. By Lemma 7.15, it does not decrease the distance between s and p. The algorithm consists of $|V|$ steps corresponding to the set of all possible distances between s and p. A step consists of a set of iterations when the distance between s and p remains unchanged.

Let us consider successive iterations that let the distance $dist(s,t)$ unchanged. During some iterations, the auxiliary network G_0 changes since some arcs are added and other arcs are removed. However, since $d(s,t)$ remains the same, the paths along which pushes are performed use only arcs of G_0. Since at each iteration, at least one arc of G_0 is removed (we push the maximum possible along the chosen path, so one arc is removed), at most $|E(G)|$ such iterations are performed.

Since $dist(s,t) \leq |V(G)|$, we have at most $|V(G)|$ steps of at most $|E(G)|$ iterations each. At each iteration we mainly have to find a shortest directed (s,t)-path. This can be done in time $O(|E(G)|)$ by any search algorithm (See Chapter 2). Hence the total complexity is $O(|E(G)|^2|V(G)|)$.

□

7.7 Algorithm using a scale factor

If the capacities are integral then Algorithm 7.1 may take time to find the maximum flow becasue of the disparity of the values of the capacities. In the example in Figure 7.8, the pushes may be performed along an arc with capacity 1, while there is a path between the source and the sink with capacity k. This problem may be overcome by using a scale factor. The idea is to try to work with capacities with similar values, to saturate them and then to consider capacities with smaller values. The idea is to start with the greatest capacities, to push some flow in a network that consists only of these links and to try to saturate them as much as possible. Then, the network is replaced by the current auxiliary network to which we add new links with smaller capacities than the one considered yet.

Algorithm 7.3 (Scaling Algorithm)**.**

0. Compute the smallest integer m such that $c(e) < 2^m$ for any arc e.
   For all $e \in E(G)$, set $c(e) = \sum_{j=0}^{m-1} 2^j c_j(e)$ with $c_j(e) \in \{0,1\}$.
1. $k := m-1$; for all $e \in E(G)$, $c'(e) := 0$ and $f(e) := 0$;
2. If $k < 0$, then terminate, else for all $e \in E(G)$, $c'(e) := c'(e) + 2^k c_k(e)$.
3. Increase as much as possible the flow f in (G,s,t,c'); $k := k-1$; go to Step 2.

Clearly, this algorithm computes a maximum flow because, at the end, all capacities are taken into account.

Let us consider the complexity of this algorithm. For this purpose, we need the following proposition the proof of which is left in Exercise 7.10.

Proposition 7.17. *Let $N = (G,s,t,c)$ be a flow network and α a positive real. Let e be an arc of G and N' be the flow network (G,s,t,c') where c' is defined by $c'(e) = c(e) + \alpha$ and $c'(f) = c(f)$ for all $f \neq e$. Then $v_{max}(N') \leq v_{max}(N) + \alpha$.*

Theorem 7.18. *Algorithm 7.3 performs less than $(\log_2(v_{max}) + 1) \cdot |E(G)|$ pushes.*

Proof. Let $c'_i = \sum_{j=i}^{m-1} 2^j c_j$ be the capacity at Step 3 when $k = i$, g_i be the flow added during this step, and f_{i+1} be the total flow before this step. Hence $f_i = f_{i+1} + g_i$.

Let us prove by decreasing induction that at Step 3-(i) Algorithm 7.3 performs at most $|E(G)|$ pushes and that $f_i(e)$ is a multiple of 2^i for every arc e.
The results holds trivially for $i = n$. Suppose now that the results holds for $i+1$. Since the capacity c'_i and the flow f^{i+1} are multiple of 2^i, in the auxiliary network, the capacity of the arcs are multiples of 2^i. Hence finding g_i corresponds to finding a maximum flow g'_i in the network with integral capacities $(G,s,t,(c'_i - f_{i+1})/2^i)$ with $v(g'_i) \leq v(g_i)/2^i$. Moreover, at the end of Step 3-($i+1$), the flow could not be increased anymore, therefore, there are no directed paths with minimum capacity 2^{i+1}. Hence, at Step 3.(i), any directed path in the auxiliary network has minimal capacity 2^i. Hence, each iteration pushes exactly 2^i units of flow at Step 3.(i). Thus $v(g_i) \leq 2^i \cdot |E(G)|$ and so $v(g'_i) \leq 2^i |E(G)|$. By Proposition 7.11, finding g'_i and g_i is done in at most $|E(G)|$ pushes. In addition, by Remark 7.12, the flow g'_i has integral values and so the values of g_i are multiple of 2^i.

Let i_0 be the largest integer such that a push has been done at i_0. If $2^{i_0} \geq v_{max}$, then after pushing once at Step 3-(i_0), we obtained a flow of value 2^{i_0}, which must be a maximum flow.

Otherwise, $i_0 < \log_2(v_{max})$ and for each i, $0 \leq i \leq i_0$, Algorithm 7.3 performs at most $|E(G)|$ pushes. Hence in total, the number of pushes it at most $(i_0+1)|E(G)| \leq (\log_2(v_{max}) + 1) \cdot |E(G)|$. □

7.8 Flows in undirected graphs

Until now, we have considered the problem in directed graphs. However, there are some contexts in which the corresponding network is undirected. For instance, when the links are bidirectional. In this case, each link has one maximum capacity but the flow may circulate in both directions if the sum of the two traffics is at most the capacity of the link.

A *undirected flow network* $N = (G, pr_{max}, co_{max}, c)$ is defined similarily to the directed case. The definition of the flow is a bit modified since two values must be associated to each edge uv: $f(u,v)$ corresponds to the traffic from u to v and $f(v,u)$ corresponds to the traffic from v to u.

Definition 7.19 (Undirected Flow)**.** The *flow* is a three-tuple $F = (pr, co, f)$ where

- pr is a function of production such that, for any vertex v, $0 \leq pr(v) \leq pr_{max}(v)$;
- co is a function of consumption such that, for any vertex v, $0 \leq co(v) \leq co_{max}(v)$;
- f is a function over the ordered pair (u,v) (with u and v adjacent), called *flow function* that satisfies the following constraints:

Positivity:	$\forall e \in E$,	$f(e) \geq 0$
Capacity constraint:	$\forall uv \in E$,	$f((u,v)) + f((v,u)) \leq c(uv)$
Flow conservation:	$\forall v \in V$,	$\sum_{(u,v)\in E} f((u,v)) + pr(v) = \sum_{(v,u)\in E} f((v,u)) + co(v)$

As in the directed case, the *value of the flow F* is

$$v(F) = \sum_{v\in V} pr(v) = \sum_{v\in V} co(v)$$

Let $N = (G, pr_{max}, co_{max}, c)$ be an undirected flow network. The (directed) flow network *associated* to N is $\vec{N} = (\vec{G}, pr_{max}, co_{max}, \vec{c})$ obtained by replacing each edge uv by an arc (u,v) and an arc (v,u) each of which has capacity $c(u,v)$. Formally, $(\vec{G} = (V(G), \bigcup_{uv\in E(G)} \{(u,v),(v,u)\})$ and $\vec{c}((u,v)) = \vec{c}((v,u)) = c(uv)$.

Clearly, a flow function f in N and a flow function $\vec{f}$ in $\vec{N}$ satisfy the same constraints but the capacity constraint. The one in N:

$$\forall uv \in E(G), f((u,v)) + f((v,u)) \leq c(uv)$$

is stronger than the one in $\vec{N}$:

$$\forall uv \in E(G), \vec{f}((u,v)) \leq c(uv) \text{ et } \vec{f}((v,u)) \leq c(uv)$$

Hence, any flow of N is a flow in $\vec{N}$. Hence, the maximum value of a flow in N is at most the maximum value of a flow in $\vec{N}$. In other words, $v_{max}(N) \leq v_{max}(\vec{N})$. We show that they are equal.

Definition 7.20 (simple flow)**.** A flow is *simple* if, for any pair of vertices $\{u,v\}$, we have either $f(u,v) = 0$ or $f(v,u) = 0$. To any flow, there is a corresponding simple flow defined as follows. If $f(u,v) \geq f(v,u) > 0$ then $\tilde{f}(u,v) = f(u,v) - f(v,u)$ and $\tilde{f}(v,u) = 0$. It is easy to see that $v(f) = v(\tilde{f})$ since $f(v,s) = 0$ for any vertex v because $d^-(s) = 0$ and so $f(s,v) = \tilde{f}(s,v)$.

Proposition 7.21. *Let N be an undirected flow network and let $\vec{N}$ be the associated flow network:*

$$v_{max}(N) = v_{max}(\vec{N})$$

Proof. Let $\vec{F} = (pr, co, \vec{f})$ be a flow in $\vec{N}$. Let $F = (pr, co, f)$ be the simple flow of $\vec{F}$. Then $v(F) = v(\vec{F})$. Since F is simple, for any edge $uv \in E(G)$, we have $f(u,v) = 0$ or $f(v,u) = 0$. Besides, the capacity constraint in $\vec{N}$ gives $f(u,v) \leq c(uv)$ ou $f(v,u) \leq c(uv)$. Hence, for any $uv \in E(G)$, $f((u,v)) + f((v,u)) \leq c(uv)$. Then, F is a flow for N.

Therefore, any flow of $\vec{N}$ corresponds to a flow of N with same value. □

This proposition allows us to reduce the undirected problem to the directed case. To find a maximum flow in an undirected network, it is sufficient to solve the problem in the associated directed network and to take the corresponding simple flow.

7.9 Applications of flows

7.9.1 Connectivity in graphs

Menger's Theorem (Theorem 5.18 is very closely related to Theorem 7.7. Observe that the arc set of an (s,t)-cut in a flow network corresponds to an (s,t)-edge-separator. Hence one can deduce Menger's Theorem from Theorem 7.7. We now do it for Theorem 5.18-(ii) for digraphs. In fact the proof of this results we gave in Section 5.6 was using the push technique (in disguise).

Proof of Theorem 5.18-(ii) for digraphs. Let G be a digraph. The edge-connectivity $\kappa'(s,t)$ is the value of the capacity of a cut in the flow network N obtained by assigning to each arc a capacity of 1, and choosing s as source and t as sink. Indeed, if $C = (V_s, V_t)$ is a cut, then after the removal of the arcs of C, there remain no (s,t)-paths. Hence $\kappa'(s,t) \leq \delta_{min}(N)$. Reciprocally, let E' be an arc-separator of G and $G' = G \setminus E'$. Let V_s be the set of vertices w of G' such that there exists a directed (s,w)-path in G', and $V_t = V \setminus V_u$. By definition of V_s, any arc (x,y) with $x \in V_s$ and $y \in V_t$ is in E'. Hence, $\delta((V_s,V_t)) \leq |E'|$. Therefore, $\delta_{min}(N) \leq \kappa'(s,t)$.

From Theorem 7.7, there is a flow of value $\kappa'(s,t)$ in this network. Moreover, by Remark 7.12, we may assume that this flow is integral. Since the edges have unit capacity, the flow of value $\kappa'(s,t)$ can be decomposed into a set of $\kappa'(s,t)$ pairwise edge-disjoint (s,t)-paths. Hence $\kappa'(s,t) = \Pi'(s,t)$. □

The other cases of Menger's Theorem may also be derived from Theorem 7.7. See Exercise 7.13. Thus Menger's Theorem can be viewed as a particular case of Theorem 7.7. In fact, they are equivalent and it is not too difficult to prove Theorem 7.7 from Menger's Theorem. See Exercise 7.14.

7.9.2 Maximum matching in bipartite graphs

The theorems on matching in bipartite graphs that we proved in Chapter 6 are direct applications of flows. Indeed to every bipartite graph $G = ((A,B),E)$, one can associate the flow network $N_G = (H,s,t,c)$ defined as follows: H is the digraph obtained from G by orienting all edges of G from A to B and adding a source s and a sink t, all arcs (s,a) for $a \in A$ and (b,t) for $b \in B$; the capacity equals 1 for all arcs. See Figure 7.9.

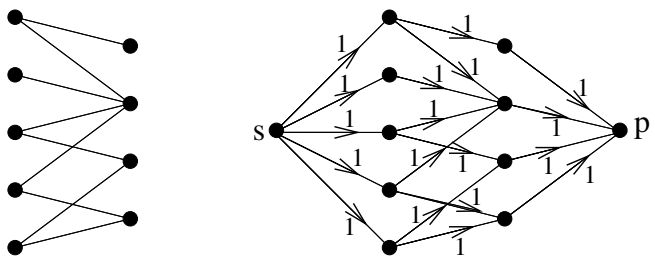


Figure 7.9: A bipartite graph and its associated flow network

There is a one-to-one correspondence between the matchings of G and the integral flows in N_G: to every matching M corresponds the flow f_M with value 1 on the arc set $\bigcup_{(a,b)\in E(G)}\{(s,a),(a,b),(b,t)\}$. Moreover the size of M is equal to the value of the flow f_M. In addition, an M-augmenting path in G corresponds to a directed (s,t)-path in the auxiliary network $N_G(f_M)$. Hence Algorithm 6.2 is a particular case of Algorithm 7.1.

One can also deduce all the results of Chapter 6 from Theorem 7.7. For example, we now give a proof of Theorem 6.3 (*"Let $G=(A,B)$ bipartite, there is a matching of size k if and only if $\forall S\subset A, |A|-|S|+|N(S)|\geq k$"*) using flows.

Proof of Theorem 6.3. Let us prove that the maximum size μ of a matching in G is equal to the maximum value of a flow in N_G. If M is a matching of size μ in G, then f_M has value μ. Hence $\mu \leq v_{max}$. Reciprocally, from Remark 7.12, there is a maximum flow f with integral values. It is in one-to-one correspondence with a matching M of size $v(f)=v_{max}$. Hence, $\mu \geq v_{max}$.

Let $C=(V_s,V_t)$ be a minimum cut. Let $A_s=A\cap V_s$ and $B_s=B\cap V_s$. If there is a vertex $b\in(B\cap N(A_s))\setminus B_s$, then setting $C'=(V_s\cup\{b\},V_t\setminus\{b\})$, we get $\delta(C')\leq\delta(C)-1+1=\delta(C)$. So, by adding the vertices of $(B\cap N(A_s))\setminus B_s$ in V_s if needed, we may assume that the minimum cut that we consider is such that $N(A_s)\subseteq B_s$.

Let us consider the capacity of C.

$$\begin{aligned}\delta_{min}=\delta(C) &= |\{(s,a)\mid a\in A\setminus A_s\}|+|\{(b,t)\mid b\in B_s\}|+|\{(a,b)\}\mid a\in A_s, b\in B\setminus B_s\}|\\ &= |A|-|A_s|+|B_s|\end{aligned}$$

Since $N(A_s)\subseteq B_s$, it follows that $\delta_{min}\geq |A|-|A_s|+|N(A_s)|$. Moreover, the cut $(A_s\cup N(A_s), V(H)\setminus(A_s\cup N(A_s)))$ has capacity $|A|-|A_s|+|N(A_s|$. So $\delta_{min}=|A|-|A_s|+|N(A_s)|$. We conclude that

$$\delta_{min}=\min\{S\subset A\mid |A|-|S|+N(S)\}$$

Hence, from Theorem 7.7, $\mu=v_{max}=\delta_{min}=\min\{S\subset A\mid |A|-|S|+N(S)\}$. □

7.9.3 Maximum-gain closure

In this problem, we have several jobs. Each job $j\in J$ is associated to a gain $g(j)$. The gain may be negative (if so, it corresponds to a loss). We note J^+ (resp. J^-) the set of jobs with positive gain (resp., negative gain).

Besides, there are several *closure* constraints.That is, the choice of a job may imply the choice of one or several other jobs. We represent this closure relation (e.g., implication relation) by an *implication digraph* D_J: its vertices are the jobs and there is an arc (j_1,j_2) if and only if the choice of j_1 implies the choice of j_2.

The objective is to find a set of *compatible*jobs with maximum gain, that is a subset A of J such that:

- there are no arcs leaving A ($E((A,\overline{A}))=\emptyset$) i.e. A is a *closure* in D_J;
- $\sum_{j\in A} g(j)$ is maximum.

Let us show how this problem can be reduced to a problem of cut with minimum capacity, hence to a maximum flow problem. Let $N = (G, s, t, c)$ be the following flow network (See Figure 7.10:

- $V(G) = T \cup \{s,t\}$;
- for any job j with positive gain, link the source s to the job j with an arc (s, j) with capacity $c(s, j) = g(j)$;
- for any job j with non-positive gain, link j to the sink t with an arc (j,t) with capacity $c(j,t) = -g(j)$;
- if a job j_1 implies a job j_2, we add the arc (j_1, j_2) with infinite capacity in the network.

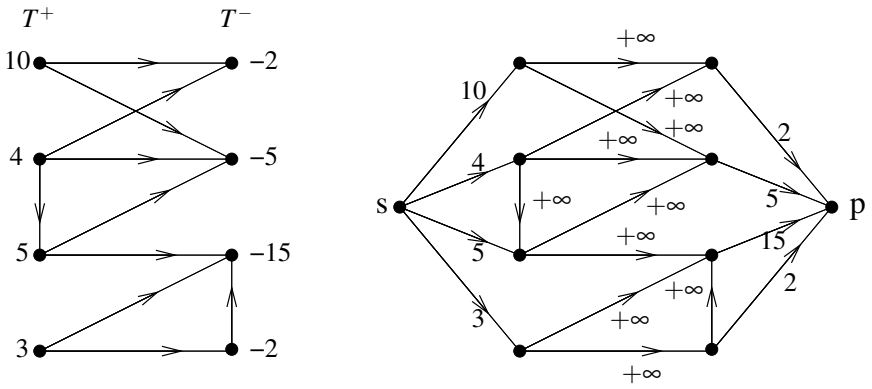


Figure 7.10: An implication digraph and its corresponding flow network

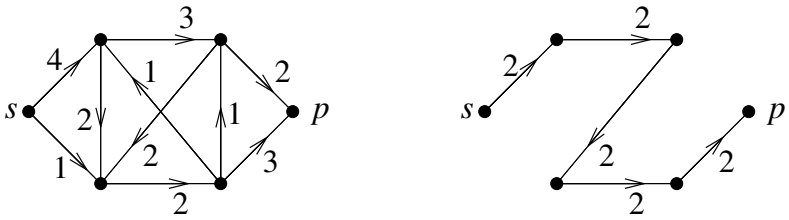
Let us consider a cut $C = (V_s, V_t)$ with finite capacity in the network. We have $S = \{s\} \cup A$ with $A \subset J$. Moreover, since the cut has finite capacity, its arc set contains no arcs of D_J, so A is a set of compatible jobs. Let $A^+ = A \cap J^+$ and $A^- = A \cap J^-$. The capacity of C is:

$$\begin{aligned}
\delta(C) &= \sum_{j \in J^+ \setminus A^+} c(s,j) + \sum_{j \in A^-} c(j,t) \\
&= \sum_{j \in J^+ \setminus A^+} g(j) - \sum_{j \in A^-} g(j) \\
&= \sum_{j \in J^+} g(j) - \sum_{j \in A^+} g(j) - \sum_{j \in A^-} g(j) \\
&= \sum_{j \in J^+} g(j) - \sum_{j \in A} g(j)
\end{aligned}$$

Since $\sum_{j \in J^+} g(j)$, the sum of all positive gains, is a constant, minimizing the capacity of the cut $S = \{s\} \cup A$ is equivalent to maximizing the gain of A.

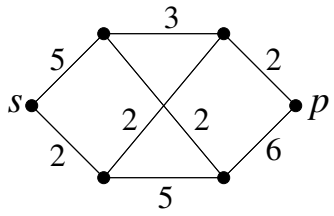
7.10 Exercices

Exercise 7.1. Let N be the flow network and f_0 the (s,t)-flow in N as depicted in the figure below.



1) Start from f_0 and find a maximum (s,t)-flow. Detail the steps of the algorithm.
2) Describe a minimum cut of this network.

Exercise 7.2. Find a maximum (s,t)-flow and a minimum (s,t)-cut in the network depicted below. (Detail the steps of the "push" algorithm.)



Exercise 7.3. There are 3 production sites A,B,C and 5 consumption sites $1,2,3,4,5$; their production and consumption, respectively, are given in the following tables.

A	B	C
5	4	7

1	2	3	4	5
3	4	5	2	1

Finally, each production sites can only serve the consumption sites as summarized in the following table.

A	B	C
13	24	345

The problem is to satisfy the consumption sites. Model the following problem in terms of flows and give a solution to the problem or explain why it could not exist.

Exercise 7.4. Prove the following property: *for all* (s,t)*-cut* $C = (V_s, V_t)$, $v(f) = out(f,C) - in(f,C)$. Why is the hypothesis $s \in V_s$ and $p \in V_t$ important?

Exercise 7.5. Prove Lemma 7.9. Verify that the positivity, the capacity constraint and the flow conservation are satisfied for f'.

Exercise 7.6. The *support* of a flow is the set of arcs on which the flow function is positive. Show that there always exists a maximum flow whose support has no directed cycle.

Exercise 7.7. Let $N = (G,s,t,c)$ be a flow network such that, for all arc e, $c(e)$ is an even integer.
1) Prove that the maximum value of a flow is an even integer.
2) Show that there is a maximum flow f such that, for all arc e, $f(e)$ is an even integer.

Exercise 7.8. Modify Algorithm 7.1 to obtain an algorithm finding a minimum cut in a flow network.

Exercise 7.9. Construct a flow network for which Algorithm 7.1 produces a sequence of flow whose values converges to a finite value when pushing on some sequence of directed paths, while the value of a maximum flow is infinite (or arbitrarily large).

Exercise 7.10. Prove Proposition 7.17.

Exercise 7.11. In this exercise, we study a variant of the algorithm for finding a maximum flow, in which we push the flow along a directed path of maximum residual capacity.

1) Give an algorithm finding a directed path of maximum residual capacity.

2) Show that if we push an amount of x at one step, then we push an amount of at least x at each following step.

Exercise 7.12. Let $N = (G,s,t,c)$ be a flow network. To each vertex $v \in V(G)$, we associate an real $w(v)$. We want to compute a flow f of maximum value satisfying the following extra constraint: $\forall v \in V(G)$ the flow entering v is at most $w(v)$ (i.e. $\sum_{u \in N^-(v)} f(uv) \leq w(v)$). Show how to find such a flow by computing a maximum flow on a network obtained from N by slight modifications.

Exercise 7.13. Deduce Menger's Theorem (5.18) from Theorem 7.7 . (*Hint*: One can use Exercise 7.12 to prove Theorem 5.18-(i).

Exercise 7.14. Deduce Theorem 7.7 from Menger's Theorem (5.18).

Exercise 7.15. Several companies send members to a conference; the ith company send m_i members. During the conference, several workshops are organized simulteanously; the ith workshops can receive at most n_j participants. The organizers want to dispatch participants into workshops so that two members of a same company are not in a same workshop. (The workshop do not need to be full.)

a) Show how to use a flow network for testing if the constraints may be satisfied.

b) If there are p companies and q workshops indexed in such a way that $m_1 \geq \cdots \geq m_p$ and $n_1 \leq \cdots \leq n_q$. Show that there exists a dispatching of participants into groupes satisfying the constraints if and only if, for all $0 \leq k \leq p$ and all $0 \leq l \leq q$, we have $k(q-l) + \sum_{j=1}^{l} n_j \geq \sum_{i=1}^{k} m_i$.

Exercise 7.16 (Unsplittable flow)**.** We consider a flow network with one production site s and many consumption sites, say $t_1, t_2, \ldots t_n$. The consumption at site t_i is d_i. We want to route commodities for s to t_i with the following additionnal constraint: the traffic from s to t_i must be routed along a unique directed path (it cannot be split).

Show that this problem is NP-complete.

Chapter 8

Graph colouring

8.1 Vertex colouring

A *(vertex) colouring* of a graph G is a mapping $c : V(G) \to S$. The elements of S are called *colours*; the vertices of one colour form a *colour class*. If $|S| = k$, we say that c is a *k-colouring* (often we use $S = \{1, \ldots, k\}$). A colouring is *proper* if adjacent vertices have different colours. A graph is *k-colourable* if it has a proper k-colouring. The *chromatic number* $\chi(G)$ is the least k such that G is k-colourable. Obviously, $\chi(G)$ exists as assigning distinct colours to vertices yields a proper $|V(G)|$-colouring. An *optimal colouring* of G is a $\chi(G)$-colouring. A graph G is *k-chromatic* if $\chi(G) = k$.

In a proper colouring, each colour class is a stable set. Hence a *k-colouring* may also be seen as a partition of the vertex set of G into k disjoint *stable sets* $S_i = \{v \mid c(v) = i\}$ for $1 \leq i \leq k$. Therefore k-colourable are also called *k-partite graphs*. Moreover, 2-colourable graphs are very often called *bipartite*.

Clearly, if H is a subgraph of G then any proper colouring of G is a proper colouring of H.

Proposition 8.1. *If H is a subgraph of G, then $\chi(H) \leq \chi(G)$.*

Proposition 8.2. $\chi(G) = \max\{\chi(C), C \text{ connected component of } G\}$.

Proof. Proposition 8.1 gives the inequality $\chi(G) \geq \max\{\chi(C), C \text{ connected component of } G\}$ because every connected component of G is a subgraph of G.

Let us now prove the opposite inequality. Let $C_1, C_2, \ldots, C_p$ be the connected components of G. For $1 \leq i \leq p$, let c_i be a proper colouring of C_i with colours $1, 2, \ldots, \chi(C_i)$. Let c be the union of the c_i that is the colouring of G defined by $c(v) = c_i(v)$ for all $v \in C_i$. Since there is no edge between two vertices in different connected component, c is a proper colouring of G with colours $1, 2, \ldots, \max\{\chi(C_i) \mid 1 \leq i \leq p\}$. Hence $\chi(G) \leq \max\{\chi(C), C \text{ connected component of } G\}$. □

Often in the following, we will consider connected graphs.

The 1-colourable graphs are the empty graphs (i.e. graphs with no edges). The 2-colourable graphs are the bipartite graphs and can be characterized and recognized in polynomial time (See Section 2.3. However for $k \geq 3$ the k-colouring problem becomes difficult: for all $k \geq 3$. it is

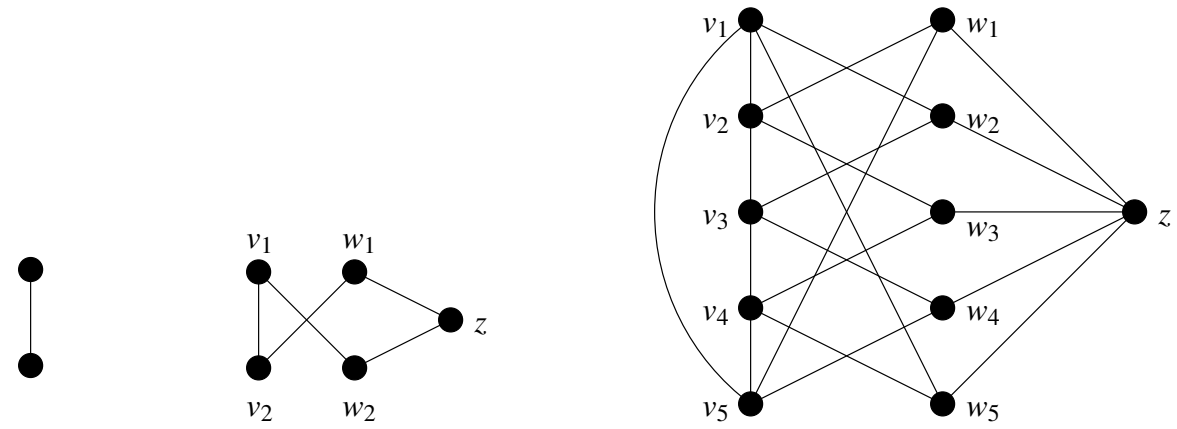


Figure 8.1: The graphs M_2, M_3 and M_4.

$\mathcal{NP}$-complete to decide if a graph is k-colourable. See [11]. Furthermore, it is $\mathcal{NP}$-hard to approximate the chromatic number within $|V(G)|^{\varepsilon_0}$ for some positive constant ε_0 as shown by Lund and Yannakakis [17].

8.1.1 Lower bounds for $\chi(G)$

Clearly, the complete graph K_n requires n colours, so $\chi(K_n) = n$. Together with Proposition 8.1, it yields the following.

Proposition 8.3. $\chi(G) \geq \omega(G)$.

This bound can be tight, but it can also be very loose. Indeed, for any given integers $k \leq l$, there are graphs with clique number k and chromatic number l. For example, the fact that a graph can be triangle-free ($\omega(G) \leq 2$) and yet have a large chromatic number has been established by a number of mathematicians including Descartes (alias Tutte) [7] (See Exercise 8.7), Kelly and Kelly [15] and Zykov [27]. We present here a proof of this fact due to Mycielski [19].

Theorem 8.4. *For every positive integer k, there exists a triangle-free k-chromatic graph.*

Proof. The *Mycielski graphs* M_k, $k \geq 1$ are defined inductively as follows. $M_1 = K_1$ and $M_2 = K_2$. For $k \geq 2$, let $V(M_k) = \{v_1, v_2, \ldots, v_n\}$. The graph M_{k+1} is defined by $V(M_{k+1}) = V(M_k) \cup \{w_1, w_2, \ldots, w_n, z\}$ and $E(M_{k+1}) = E(M_k) \cup \{w_i v_j, v_i v_j \in E(M_k)\} \cup \{w_i z, 1 \leq i \leq n\}$. See Figure 8.1.

Let us show by induction on $k \geq 1$ that M_{k+1} is triangle-free and k-chromatic, the result holding trivially for $k = 1$.

Let us first show that it is triangle-free. $W = \{w_1, w_2, \ldots, w_n\}$ is a stable set of M_{k+1} and z is adjacent to no vertex of M_k. So z is in no triangle. In addition, if there is a triangle T in M_{k+1}, then two of the three vertices must belong to M_k and the third vertex must belong to W, say

$V(T) = \{w_i, v_j, v_k\}$. Since w_i is adjacent to v_j and v_k, by definition of M_{k+1} it follows that v_i is also adjacent to v_j and v_k. Hence v_i, v_j and v_k induce a triangle in M_k, which is a contradiction. Thus, as claimed, M_{k+1} is triangle-free.

Next, we show that $\chi(M_{k+1}) = k+1$. Since M_k is a subgraph of M_{k+1}, by Proposition 8.1, $\chi(M_{k+1}) \geq k$. Let a k-colouring of H in $\{1,\ldots,k\}$ be given. Assign to w_i the same colour that is assigned to v_i for $1 \leq i \leq n$ and assign $k+1$ to z. The obtained colouring is a proper $(k+1)$-colouring of M_{k+1} and so $\chi(M_{k+1}) \leq k+1$. Suppose that $\chi(M_{k+1}) = k$. Then there is a k-colouring of M_{k+1} with colours $\{1,\ldots,k\}$. Without loss of generality, we may ssume that z is coloured k. Then no vertex of W is coloured k. For each vertex v_i of M_k coloured k, recolour it with the colour assigned to w_i. Because the neighbours of v_i are also neighbours of w_i, this produces a proper colouring of M_k. Moreover this colouring uses $k-1$ colours. This is a contradiction, thus $\chi(M_{k+1}) \geq k+1$ and so $\chi(M_{k+1}) = k+1$. □

Proposition 8.5. $\chi(G) \geq \frac{|V(G)|}{\alpha(G)}$.

Proof. Let c be a proper colouring of G with colours $1,2,\ldots,\chi(G)$. For $1 \leq i \leq \chi(G)$, let S_i be the stable set of vertices coloured i. Then $|S_i| \leq \alpha(G)$. So

$$|V(G)| = \sum_{i=1}^{\chi(G)} |S_i| \leq \sum_{i=1}^{\chi(G)} \alpha(G) \leq \chi(G).\alpha(G).$$

Hence $\chi(G) \geq \frac{|V(G)|}{\alpha(G)}$. □

Again this bound can be very loose. For example, consider a graph G with n connected components all of which are isomorphic to K_1 except one which is isomorphic to K_k. Then $\chi(G) = k$ and $\frac{|V(G)|}{\alpha(G)} = \frac{n+k-1}{n}$ which is less than 2 for n sufficiently large.

8.2 Chromatic number and maximum degree

Most upper bounds on the chromatic number come from algorithms that produce colourings. The most widespread one is the greedy algorithm. A *greedy colouring* relative to a vertex ordering $(v_1 < \cdots < v_n)$ of $V(G)$ is obtained by colouring the vertices in the order $v_1,\ldots,v_n$, assigning to v_i the smallest-indexed colour not already used on its lower-indexed neighbourhood. In a vertex-ordering, each vertex has at most $\Delta(G)$ earlier neighbours, so the greedy colouring cannot be forced to use more than $\Delta(G)+1$ colours.

Proposition 8.6. $\chi(G) \leq \Delta(G)+1$.

The bound $\Delta(G)+1$ is the worst number of colours that a greedy colouring can have. However there is a vertex ordering whose associated colouring is optimal colouring. Indeed, if c is an optimal colouring of G, then any ordering $\sigma_{opt} = (v_1 < \cdots < v_n)$ such that for any $i < j$, $c(v_i) \leq c(v_j)$ will be. But finding such an ordering among the $n!$ possible orderings is difficult because it is $\mathcal{NP}$-hard to determine the chromatic number of a graph.

The bound $\Delta(G)+1$ may be lowered by finding orderings yielding a greedy colouring with less than $\Delta(G)+1$ colours. A graph G is *k-degenerate* if each of its subgraphs has a vertex of degree at most k. The *degeneracy* of G, denoted $\delta^*(G)$, is the smallest k such that G is k-degenerate. It is easy to see that a graph is k-degenerate if and only if there is an ordering $(v_1 < v_2 < \cdots < v_n)$ of the vertices such that for every $1 < i \leq n$, the vertex v_i has at most k neighbours in $\{v_1, \ldots, v_{i-1}\}$. Hence the greedy colouring relative to this ordering uses at most $\delta^*(G)+1$ colours.

Proposition 8.7.

$$\chi(G) \leq \delta^*(G)+1.$$

Note that finding an ordering as above (and thus the degeneracy of a graph) is easy. It suffices to recursively take a vertex v_n of minimum degree in the graph and to put it at the end of the ordering $v_1, \ldots, v_{n-1}$ of $G - v_n$.

Proposition 8.8. *Let G be a connected graph. Then $\delta^*(G) = \Delta(G)$ if and only if G is regular.*

Proof. Assume first that G is regular. Then for all ordering the last vertex has $\Delta(G)$ lower indexed neighbours. So $\delta^*(G) = \Delta(G)$.

Assume now that G is not Δ-regular. Let v_n be a vertex of degree less than Δ. Since G is connected, one can grow a spanning tree of G from v_n, assigning indices in decreasing order as we reach vertices. We obtain an ordering $v_1 < \cdots < v_n$ such that every vertex other than v_n has a higher-indexed neighbour. Hence $\delta^*(G) < \Delta(G)$. □

This proposition and Proposition 8.6 implies the following.

Corollary 8.9. *Let G be a connected graph. If G is not regular, then $\chi(G) \leq \Delta(G)$.*

In view of this corollary, one may wonder which connected graphs G satisfies $\chi(G) = \Delta(G)$. It is the case for complete graphs. On can also easily see that it is also the case for odd cycles. Brooks showed [6] that they are the only ones.

Theorem 8.10 (BROOKS' THEOREM)**.** *Let G be a connected graph. Then $\chi(G) \leq \Delta(G)$ unless G is either a complete graph or an odd cycle.*

In order to prove this theorem, we need the following preliminary results.

Proposition 8.11. *Let G be a connected graph which is not a complete graph. Then there exists three vertices u, v and w such that $uv \in E(G)$, $vw \in E(G)$ and $uw \notin E(G)$.*

Proof. Since G is not complete, there exists two vertices u and u' which are not linked by an edge. Because G is connected, there is a path between u and u'. Let P be a shortest (u,u')-path and let v and w be respectively the second and third vertices on P. Then uv and vw are edges of the paths and uw is not an edge for otherwise it would shortcut P. □

Lemma 8.12. *Let G be a 2-connected graph of minimum degree at least 3. If G is not a complete graph, then there exists a vertex x having two non-adjacent neighbours v_1 and v_2 such that $G - \{v_1, v_2\}$ is connected.*

Proof. If G is 3-connected, then the results follows directly from Proposition 8.11.

So we may assume that G is not 3-connected. Hence it as a separator of size 2, say $\{x,y\}$. Since G is 2-connected, then $G-x$ is connected. Moreover $G-x$ has at least one separating vertex y and this has two end-blocks B_1 and B_2. (See Exercise 5.27). Now, since G is 2-connected, for $i=1,2$, the separating vertex y_i in B_i is not a separating vertex of G and thus x is adjacent to a vertex v_i of $B_i \setminus \{y_i\}$. Hence by Exercise 5.27 4), $G-\{x,v_1,v_2\}$ is connected. But x has degree at least 3 in G, and so has a neighbour in $V(G)\setminus\{v_1,v_2\}$. Therefore, $G-\{v_1,v_2\}$ is connected. □

Proof of Theorem 8.10. If G is not regular, Corollary 8.9 yields the result. So we may assume that G is regular. In addition, we may assume that $\Delta=\Delta(G)\geq 3$, since G is complete if $\Delta\leq 1$ and G is a cycle when $\Delta=2$, in which cases the result holds.

We shall find an ordering of the vertices so that the greedy colouring relative to it yields the desired bound.

Assume now that G is Δ-*regular*. If G has a cut-vertex x. Let $C_1,C_2,\ldots,C_p$ be the connected components of $G-x$. For $1\leq i\leq p$ let G_i be the graph induced by $V(C_i)\cup\{x\}$. Each of these graphs is not regular because x has degree less than Δ. So by Corollary 8.9, all the G_i, $1\leq i\leq p$ have a proper Δ-colouring. Free to permute the colours, one can assume that the colourings agree on x. Then the union of these colourings is a Δ-colouring of G.

Hence we may assume that G is 2-connected. In such a case, for G is not complete, by Lemma 8.12 some vertex v_n has two non-adjacent neighbours v_1 and v_2 such that $G-\{v_1,v_2\}$ is connected. Then indexing the vertices of a spanning tree of $G-\{v_1,v_2\}$ rooted in v_n in a decreasing order, with $\{3,\ldots,n\}$, we obtain an ordering $v_1,\ldots,v_n$ such that every vertex other than v_n has a higher-indexed neighbour. Now the greedy algorithm will assign colour 1 to both v_1 and v_2. So when colouring v_n at most $\Delta-1$ colours will be assigned to its neighbours. Hence the greedy colouring will use at most Δ colours. □

Note that the above proof is constructive and yields a polynomial-time algorithm for finding a $(\Delta(G)+1)$-algorithm of a graph which is neither a complete graph nor an odd cycle.

Brooks' Theorem states that for $\Delta(G)>2$, $\chi(G)=\Delta(G)+1$ if and only if G contains a clique of size $\Delta(G)+1$. It is natural to ask whether this extends further. E. g. if $\chi(G)\geq\Delta+1-k$ does G contain a large clique? One cannot expect a clique of size $\Delta+1-k$ if k is large. Indeed consider the graph $H_{\Delta,p}$ formed by adding all the edges between a $(\Delta+1-5p)$-clique and p disjoint 5-cycles. It is easy to see that $H_{\Delta,p}$ has maximum degree Δ, chromatic number $\Delta+1-2p$ and clique number $\Delta+1-3p$. Reed [20] conjectured that if $\chi(G)\geq\Delta+1-k$ then G contains a clique of size at least $\Delta+1-2k$.

Conjecture 8.13 (REED'S CONJECTURE)**.** Let G be a graph.
If $\chi(G)\geq\Delta(G)+1-k$, then $\omega(G)\geq\Delta(G)+1-2k$. In other words,

$$\chi(G)\leq\left\lceil\frac{\Delta(G)+1+\omega(G)}{2}\right\rceil.$$

Note that this value $2k$ is best possible. Indeed consider random graph R on n vertices with edge probability $(1-n^{-3/4})$. The expected number of cliques of size i is

$$\begin{aligned}\binom{n}{i}\left(1-n^{-3/4}\right)^{\binom{i}{2}} &\leq 2^{i\log n}\left(1-n^{-3/4}\right)^{\frac{i^2}{4}}\\ &\leq 2^{i\log n}e^{-n^{-3/4}\frac{i^2}{4}}.\end{aligned}$$

For $i > n^{3/4}\log n$, this is $o(1)$ so (with high probability) $\omega(R) \leq n^{3/4}\log n$. Now the expected number of stable sets of size 3 is $\binom{n}{3} \times \left(n^{-3/4}\right)^3 = O(n^{3/4})$. Hence removing one vertex per such stable set, we obtain a graph H with $n - O(n^{3/4})$ vertices and stability number $\alpha(H) = 2$. Hence its chromatic number is at least $n/2 - O(n^{3/4})$. Let G be the graph obtained by connecting all the vertices of H to a clique of size $\Delta - n$. Then $\Delta(G) = \Delta$, $\chi(G) = \Delta - n + \chi(H) \geq \Delta - n/2 - O(n^{3/4})$ and $\omega(G) \leq \Delta - n + \omega(G) \leq \Delta - n + n^{3/4}\log n$.

As an evidence for Conjecture 8.13, Reed [20] showed that there is an $\varepsilon > 0$ such that $\chi(G) \leq \varepsilon\omega(G) + (1-\varepsilon)(\Delta(G)+1)$. Johannson [14] settled Conjecture 8.13 for $\omega = 2$ and Δ sufficiently large. In fact, he proved that there is a constant c such that if $\omega(G) = 2$ then $\chi(G) \leq c\frac{\Delta(G)}{\log\Delta(G)}$. Johannson's proof uses the probabilistic method and needs a careful probabilistic analysis. The interested reader is referred to Chapter 13 of [18]. However, one can easily improve the bound of Brooks' Theorem for triangle-free graphs.

Proposition 8.14. *Let G be a triangle-free graph. Then $\chi(G) \leq 3\left\lceil\frac{\Delta(G)+1}{4}\right\rceil$.*

Proof. Set $k = \left\lceil\frac{\Delta(G)+1}{4}\right\rceil$. Let $(V_1, V_2, \ldots, V_k)$ be the partition of $V(G)$ in k sets such that the number of internal edges (i.e. with two endvertices in a same part) is minimum. For all i, the graph G_i induced by V_i has maximum degree at most 3. Indeed, suppose that a vertex x has 4 neighbours in the part it belongs to, say V_1. Then there is another part, say V_2, in which x has at most 3 neighbours, otherwise x would have at least $4k \geq \Delta(G)+1$ neighbours, which is impossible. Thus the partition $(V_1 - x, V_2 + x, \ldots, V_k)$ has less internal edges than $(V_1, V_2, \ldots, V_k)$ which contradicts the minimality of this.

Hence $\Delta(G_i) \leq 3$ and $\omega(G_i) \leq 2$ as it is a subgraph of G. Hence by Brooks' Theorem, $\chi(G_i) \leq 3$.

So colouring properly each G_i with the colour set $\{3i-2, 3i-1, 3i\}$, we obtain a proper $3k$-colouring of G. □

When $k = 1$ Conjecture 8.13 asserts that if $\chi(G) = \Delta(G)$ then $\omega(G) \geq \Delta - 1$. In fact, Reed [21] showed that when Δ is large if $\chi(G) = \Delta(G)$ then $\omega(G) = \Delta(G)$, thus settling a conjecture of Beutelspacher and Hering [4]. Borodin and Kostochka [5] conjectured that it is true for $\Delta \geq 9$; counterexamples are known for each $\Delta \leq 8$.

Conjecture 8.15 (Borodin and Kostochka [5])**.** Let G be a graph of maximum degree $\Delta \geq 9$. If $\chi(G) = \Delta$, then $\omega(G) = \Delta$.

When $k = 2$, one cannot expect all $(\Delta-1)$-chromatic graphs to have a clique of size $\Delta-1$. Indeed $H_{\Delta,1}$ has chromatic number $\Delta-1$ but clique number $\Delta-2$. However, Farzad, Molloy and Reed [10] showed that for Δ sufficiently large if $\chi(G) \geq \Delta-1$ then G contains either a $(\Delta-1)$-clique or $H_{\Delta,1}$. They also proved similar results for $k = 3$ and $k = 4$; in these cases, G must contain one of five or thirty eight graphs respectively.

Let k_Δ the maximum integer such that $(k+1)(k+2) \leq \Delta$. Thus, $k_\Delta \approx \sqrt{\Delta} - 2$. Molloy and Reed [18] showed that k_Δ is a threshold to Brooks-like theorems. Indeed if $k < k_\Delta$ then, if Δ is large enough, if $\chi(G) \geq \Delta - k + 1$ then G must contain a graph H that is close to a $(\Delta+1-k)$-clique, in that H has small size ($|H| \leq \Delta+1$) and cannot be $(\Delta-k)$-coloured. As a consequence, one can check polynomially if $\chi(G) \geq \Delta-k$ or not. On the opposite, if $k < k_\Delta$, then there are arbitrarily large $(\Delta+k-1)$-critical graphs (i.e $(\Delta+1-k)$-chromatic graphs such that every proper subgraph is $(\Delta-k)$-colourable) with maximum degree Δ. Furthermore, Embden-Weinert, Hougardy and Kreuter [8], proved that for any constant Δ and $\Delta-3 \leq k < k_\Delta$, determining whether a graph of maximum degree Δ is $(\Delta-k)$-colourable is $\mathcal{NP}$-complete.

8.3 Colouring planar graphs

A graph is *embeddable* on a surface Σ if its vertices can be mapped onto distincts points of Σ and its edges onto simple curves of Σ joining the points onto which its endvertices are mapped, so that two edge curves do not intersect except in their common extremity. A face of an embedding $\tilde{G}$ of a graph G is a component of $\Sigma \setminus \tilde{G}$. We denote by $F(\tilde{G})$ the set of faces of $\tilde{G}$. A graph is *planar* if it can be embedded in the plane.

Let $\tilde{G}$ be an embedding of a planar graph G. Its numbers of vertices, faces and edges are related by Euler's Formula:

$$|V(\tilde{G})| + |F(\tilde{G})| - |E(\tilde{G})| = 1 + comp(G)$$

where $comp(G)$ is the number of connected components of G.

Proof. We prove of Euler's Formula by induction on the number of edges of G.

If G has no edges, then every vertex is a connected component and the graph has a unique face, the outer one.

Suppose now that G is a planar graph on at least one edge and that the result holds for planar graphs with less edges. Let e be an edge of G. Then two cases may occur.

Assume first that e is a bridge (i.e. $G \setminus e$ has one more component than G). Then e is incident to a unique face in G. So $G \setminus e$ has as many faces as G. By the induction hypothesis, $|V(\tilde{G} \setminus e)| + |F(\tilde{G} \setminus e)| - |E(\tilde{G} \setminus e)| = 1 + comp(G \setminus e)$. So $|V(\tilde{G})| + |F(\tilde{G})| - (|E(\tilde{G})| - 1) = 1 + comp(G) + 1$.

Assume now that e is not a bridge. Then $G \setminus e$ has the same number of components as G. Then e is incident to two faces in G. Removing e transform these two faces into a single one (their union). So $G \setminus e$ has as many faces as G. By the induction hypothesis, $|V(\tilde{G} \setminus e)| + |F(\tilde{G} \setminus e)| - |E(\tilde{G} \setminus e)| = 2 - comp(G \setminus e)$. So $|V(\tilde{G})| + (|F(\tilde{G})|) - 1 - (|E(\tilde{G})| - 1) = 1 + comp(G)$. □

Corollary 8.16. *If G is a planar graph, then*

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

Proof. Let $\tilde{G}$ be an embedding of G. Every face de $\tilde{G}$ contains at least three edges and every edge is in at most two faces. Hence, considering the number N of edge-face incidences, we have $2|E(G)| \geq 3|F(\tilde{G})|$. Putting this inequality into Euler's Formula we obtain $|V(G)| + 2|E(G)|/3 \geq |E(G)| + 2$ so $3|V(G)| - 6 \geq |E(G)|$. □

Corollary 8.17. *Every planar graph has a vertex of degree at most* 5.

Proof. Let G be a planar graph. By Corollary 8.16, $\sum\{d(v) : v \in G\} = 2|E(G)| \leq 6|V(G)| - 12$. The minimum degree of G is less or equal to the the average degree which is equal to $\frac{6|V(G)|-12}{|V(G)|} < 6$. Hence there is a vertex of degree less than 6. □

Corollary 8.18. *Every planar graph is* 6*-colourable.*

Proof. Let G be a planar graph. Every subgraph of G is planar and so has minimum degree at most 5 by Corollary 8.17. Hence G is 5-degenerate. Thus, by Proposition 8.7, $\chi(G) \leq 6$. □

Theorem 8.19. *Every planar graph is* 5*-colourable.*

Proof. By induction on the number of vertices of G, the result holding trivially if G has one vertex. By Corollary 8.17, there is a vertex v of degree at most 5 in G By the induction hypothesis, the graph $G - v$ is 5-colourable. Let c be a proper 5-colouring of $G - v$. From c, we will construct a proper 5-colouring of G.

Assume first, that one of the colours, say i, is assigned to no neighbours of v. Then one can extend c by setting $c(v) = i$. (Note that this is the case if $d(v) \leq 4$.)

Hence we may assume that that v has five neighbours coloured differently. Let v_1, v_2, v_3, v_4 and v_5 be these neighbours in counter-clockwise order around v. Free to permute the colours, we may suppose that $c(v_i) = i$ for all $1 \leq i \leq 5$.

Let $C_{1,3}$ be the component of v_1 in the subgraph G induced by the vertices coloured 1 or 3. If v_3 is not in $C_{1,3}$, then interchanging the colours 1 and 3 in $C_{1,3}$ and colouring v with 1, we obtain a proper 5-colouring of G. If $v_3 \in C_{1,3}$, then there exists a path P linking v_1 to v_3 in C_1. Together wiht vv_1 anf vv_3 it forms cycle C which separates v_2 and v_4. Thus the component $C_{2,4}$ of v_2 in the subgraph of G induced by the vertices coloured 2 and 4 does not contain v_4, otherwise an edge of the path joining v_2 to v_4 in$C_{2,4}$ would cross an edge of C. Hence on can interchange the colours 2 and 4 in $C_{2,4}$ and colour v with 2 to obtain a proper 5-colouring of G. □

Theorem 8.19 is not best possible: the celebrated Four Colour Theorem by Appel and Haken [1, 2, 3] states that every planar graph is 4-colourable. A simpler proof was presented by Robertson, Sanders, Seymour and Thomas [22, 23]. However it still uses complicated reductions to a huge number of configurations (more than six hundreds) which need to be solved by computer assistance.

Theorem 8.20 (Appel and Haken [1, 2, 3])**.** *Every planar graph is* 4*-colourable.*

Remark that the proof of Theorem 8.19 does not work for 4-colouring. Indded if we are in the configuration depicted in Figure 8.2, for every pair of colours (i, j), a vertex coloured i and a vertex coloured j are in the same component in the subgraph induced by the vertices coloured i and j.

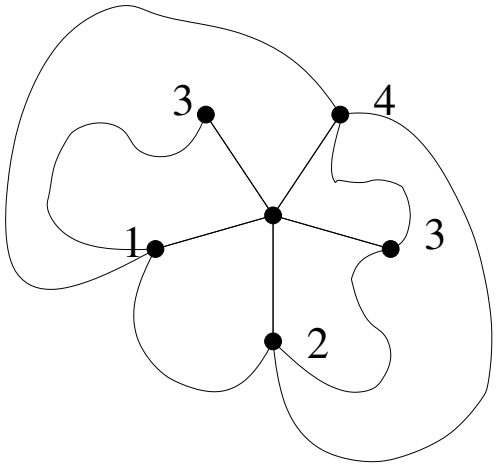


Figure 8.2: The probematic configuration. The curve from i to j represents a path whose vertices are alternately coloured i and j.

It is $\mathcal{NP}$-Complete (see [11]) to decide if the chromatic number of a planar graph is 3 or 4, even if the maximum degree does not exceed 4.

8.4 Edge-colouring

An *edge-colouring* of G is a mapping $f : E(G) \to S$. The element of S are *colours*; the edges of one colour form a *colour class*. If $|S| = k$ then f is a *k-edge-colouring*. An edge-colouring is *proper* if incident edges have different colours; that is, if each colour class is a matching. A graph is *k-edge-colourable* if it has a proper k-edge-colouring. The *chromatic index* or *edge-chromatic number* $\chi'(G)$ of a graph G is the least k such that G is k-edge-colourable.

Since edges sharing an endvertex need different colours, $\chi(G) \geq \Delta(G)$. Furthermore if a subgraph H of G is odd then a matching contains at most $\frac{|V(H)|-1}{2}$ edges. Hence at least $\frac{2|E(H)|}{|V(H)|-1}$ colours are needed to edge-colour H and thus G. It follows that

$$\chi'(G) \geq \max\left\{\Delta(G), \max\left\{\frac{2|E(H)|}{|V(H)|-1} \mid H \text{ odd subgraph of } G\right\}\right\}. \tag{8.1}$$

Observe that for any H, $\frac{2|E(H)|}{|V(H)|-1} = \frac{\sum_{v\in V(H)} d_H(v)}{|V(H)|-1} \leq \frac{|V(H)|\times\Delta(H)}{|V(H)|-1} \leq \Delta(H)+1 \leq \Delta(G)+1$.

As an edge is incident to at most $2\Delta(G)-2$ other edges ($\Delta-1$ at each endvertex), colouring the edges greedily we use at most $2\Delta(G)-1$ colours. However, one needs less colours. Vizing [25] and Gupta [12] independently showed that $\chi'(G) \leq \Delta(G)+1$.

Theorem 8.21 (Vizing [25], Gupta [12]). *If G is a graph, then $\chi'(G) \leq \Delta(G)+1$.*

Proof. We prove the result by induction on $|E(G)|$. For $|E(G)| = 0$, it is trivial.

Suppose now that $|E(G)| \geq 1$ and that the assertion holds for graphs with fewer edges than G. Set $\Delta(G) = \Delta$.

Let xy_0 be an edge of G. By induction hpothesis, $G \setminus xy_0$ admits a $(\Delta+1)$-edge-colouring. As y_0 is incident to at most $\Delta-1$ edges in $G \setminus xy_0$, there exists a colour $c_1 \in \{1,2,\ldots,\Delta+1\}$ *missing* at y_0, i.e. such that no edge incident to y_0 is coloured c_1. If c_1 is also missing at x, then colouring xy_0 with c_1, we obtain a $(\Delta+1)$-edge-colouring of G. So we may assume that there is an edge xy_1 coloured c_1.

Because y_1 is incident to at most Δ edges, a colour $c_2 \in \{1,2,\ldots,\Delta+1\}$ is missing at y_1. If c_2 is missing at x then recolouring xy_1 with c_2 and colouring xy_0 with c_1, we obtain a $(\Delta+1)$-edge-colouring of G. So we may assume that there is an edge xy_2 coloured c_2.

And so on, we construct a sequence $y_1, y_2, \ldots$ of neighbours of x and a sequence of colours $c_1, c_2, \ldots$ such that: xy_i is coloured c_i and c_{i+1} is missing at y_i. Since the degree of x is bounded, there exists a smallest l such that for an integer $k < l$,$c_{l+1} = c_k$.

Now, for $0 \leq i \leq k-1$, let us recolour the edge xy_i with c_{i+1}.

There exists a colour $c_0 \in \{1,2,\ldots,\Delta+1\}$ missing at x. In particular, $c_0 \neq c_k$. Let P be the maximal path starting at y_{k-1} with edges alternatively coloured c_0 and c_k. Let us interchange the colour c_0 and c_k on $P + xy_{k-1}$. If P does not contain y_k, we have a $(\Delta+1)$-edge-colouring of G. If P contains (and thus ends in) y_k, recolouring the edge xy_i with c_{i+1} for $k \leq i \leq l$, we obtain a $(\Delta+1)$-edge-colouring of G. □

Hence $\chi'(G) \in \{\Delta(G), \Delta(G)+1\}$. A graph is said to be *Class 1* if $\chi'(G) = \Delta(G)$ and *Class 2* if $\chi'(G) = \Delta(G)+1$. Holyer [13] showed that determining whether a graph is Class 1 or Class 2 is $\mathcal{NP}$-complete. While we will see many graphs of Class 1 and many graphs of Class 2, it turns out that it is much more likely that a graph is of Class one. Erdős and Wilson [9] proved the following, where the set of graphs of order n is denoted by $\mathcal{G}_n$ and the set of graphs of order n and of Class 1 is denoted by $\mathcal{G}_n^1$.

Theorem 8.22 (Erdős and Wilson [9]). *Almost every graph is of Class 1, that is,*

$$\lim_{n\to\infty} \frac{|\mathcal{G}_n^1|}{|\mathcal{G}_n|} = 1.$$

However there are classes of graphs for which we know if they are Class 1 or Class 2. For example, a regular graph of odd order, say $2n+1$, is Class 2 by Equation 8.1.

The following theorem of König [16] states that every bipartite graph is Class 1.

Theorem 8.23 (König [16]). *Let G be a bipartite graph. Then $\chi'(G) = \Delta(G)$.*

Proof. By induction of the number of edges, the result holding vacuously when $|E(G)| = 0$. Assume now that $|E(G)| \geq 1$. Set $\Delta = \Delta(G)$. Let xy be an edge of G. By the induction hypothesis, $G \setminus xy$ admits a proper Δ-edge-colouring.

In $G \setminus xy$, the vertices x and y are each incident to at most $\Delta - 1$ edges. So there exists c_x and c_y in $\{1, 2, \ldots, \Delta\}$ such that x (resp. y) is not incident to an edge coloured c_x (resp. c_y). If $c_x = c_y$ then assigning this colour to xy we obtain a Δ-edge-colouring of G. Hence we may assume that $c_x \neq c_y$ and that the vertex x is incident to an edge e coloured c_y.

Extend this edge into a maximal path P whose edges are coloured c_y and c_x alternatively. We claim that y is not on P. Indeed if it would be on P, since y is adajcent to no edge c_y, it would be an endvertex of P and P would terminate with an edge coloured c_x. Then $P \cup xy$ would be an odd cycle which is a contradicts Theorem 2.5. Hence one can invert the colours on P. By maximality of P, the edge-colouring is still proper. Then assigning xy the colour c_y, we obtain a Δ-edge-colouring of G. □

Planar graphs with sufficiently large maximum degree Δ are Class 1. Sanders and Zhao [24] showed that planar graphs with maximum degree $\Delta \geq 7$ are Class 1. Vizing edge-colouring conjecture [26] asserts that planar graphs of maximum degree 6 are also Class 1. This would be best possible as for any $\Delta \in \{2, 3, 4, 5\}$, there are some planar graphs with maximum degree Δ which are Class 2 [26]. However, for $\Delta \in \{3, 4, 5\}$ the complexity of deciding if a planar graph with maximum degree Δ is Δ-edge-colourable is still unknown.

Some planar cubic graphs are Class 2 as they have no perfect matching. An example is given Figure 8.3. However, by Petersen Theorem, cubic graphs with no perfect matching are

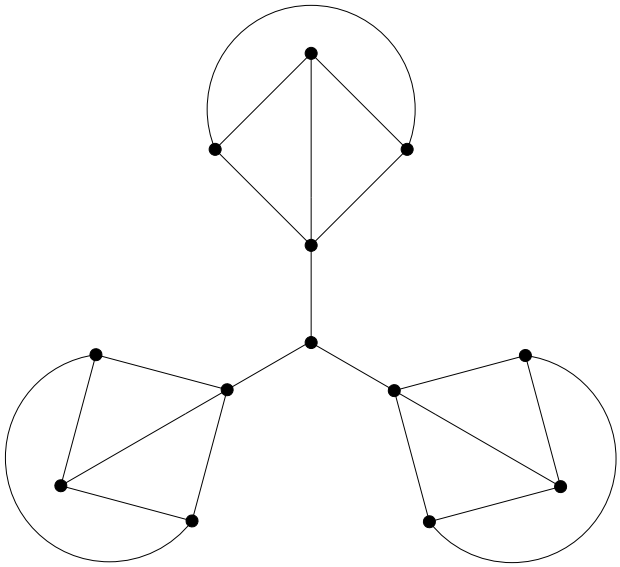


Figure 8.3: Planar cubic graph without perfect matching

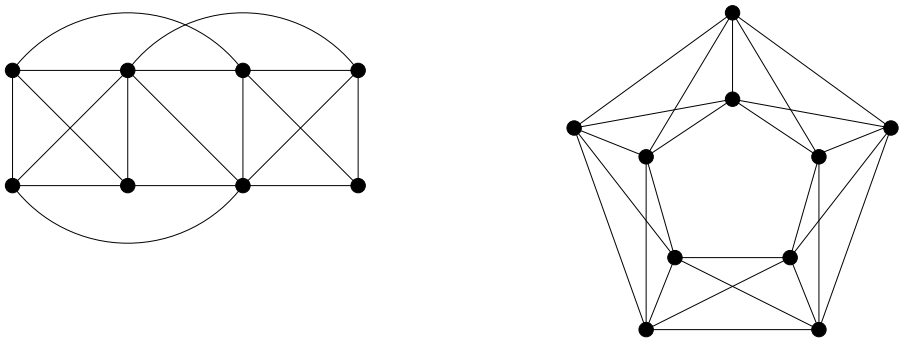
bridgeless.

Proposition 8.24. *Every bridgeless cubic planar graph is* 3*-edge colourable.*

Proof. Let G be a bridgeless cubic planar graph and $\tilde{G}$ be one of its embedding. The *dual* of $\tilde{G}$ is the graph G^* with vertex set $F(\tilde{G})$ such that two faces of $\tilde{G}$ are adjacent in G^* if they are incident to a common edge. It is easy to see that G^* is planar. So, by Theorem 8.20, G^* is 4-colourable, so one can colour the faces of G with $\{1,2,3,4\}$ such that two faces sharing an edge have different colours. For any $1 \leq i < j \leq 4$, let $E_{i,j}$ be the set of edges incident to a face coloured i and one coloured j. Observe that since G has no bridge, every edge is incident to two distinct faces. Then $E_{1,2} \cup E_{3,4}$, $E_{1,3} \cup E_{2,4}$ and $E_{1,4} \cup E_{2,3}$ are the three matchings corresponding to a proper 3-edge colouring of G. □

8.5 Exercises

Exercise 8.1. Find the chromatic number of the following graphs :



Exercise 8.2. Let G be a k-regular bipartite graph. Show that for every proper 2-colouring of G, there are as many vertices coloured 1 as vertices coloured 2.

Exercise 8.3. Show that a graph $G = (V,E)$ is 2^k-colourable if and only if E may be partitionned into k sets $E_1, \ldots, E_k$ such that for every $1 \leq i \leq k$, (V,E_i) is a bipartite graph.

Exercise 8.4. Show that if a k-chromatic graph G admits a proper colouring for which every colour is assigned to at least 2 vertices then G has a proper k-colouring with the same property.

Exercise 8.5. Let c be a partial proper colouring of a graph G with $\Delta(G) - k$ colours such that for every non-coloured vertex at least $k+1$ colours appears at least twice on its neighbourhood. Show that c can be extended to a proper $(\Delta(G) - k)$-colouring of G.

Exercise 8.6. Let G be a graph and $\overline{G}$ its complement.
1) Let v be a vertex of G.

a) Show that $\chi(G) \leq \chi(G - v) + 1$.

b) Show that if $d_G(v) < \chi(G - v)$, then $\chi(G) = \chi(G - v)$.

c) Deduce that $\chi(G) + \chi(\overline{G}) \leq |V(G)| + 1$.

2) a) Show that $\chi(G) \times \chi(\overline{G}) \geq |V(G)|$.

b) Deduce that $\chi(G)+\chi(\overline{G}) \geq 2\sqrt{|V(G)|}$.

Exercise 8.7. Let $(G_i), i \geq n$, be the sequence of graphs defined as follows : G_3 is the cycle on 5 vertices. Suppose now that G_k has n_k vertices. Set $m_k = k(n_k-1)$. Let W be a set of m_k vertices and for every subset U of W of cardinality n_k, let G_U be a copy of G_k such that W and all $V(G_U)$ are pairwise disjoints. The graph G_{k+1} is then obtained by adding for all $U \subset W$ of cardinality n_k a perfect matching between U and $V(G_U)$. Thus we have $|V(G_{k+1})| = n_{k+1} = \binom{m_k}{n_k} n_k + m_k$. Show that for all k, G_k is triangle-free and $\chi(G_k) = k$.

Exercise 8.8. A *cograph* is a graph with no subgraph isomorphic to P_4 the path on 4 vertices. Show that for any ordering of the vertices σ, the greedy algorithm produces an optimal proper colouring. (*Hint:* Suppose that the greedy algorithm uses k colours according to the ordering $(v_1 < \cdots < v_n)$ and let i be the least integer such that there is a clique formed by $k-i+1$ vertices assigned coloured from i to k. Show that $i = 1$.)

Exercise 8.9. Let G_1 and G_2 be two disjoint graphs, $x_1y_1 \in E(G_1)$ and $x_2y_2 \in E(G_2)$. The *Hajós sum* $G = (G_1, x_1y_1) + (G_2, x_2y_2)$ is the graph obtained from $G_1 \cup G_2$ by identifying x_1 and x_2, deleting x_1y_1 and x_2y_2, and adding y_1y_2.

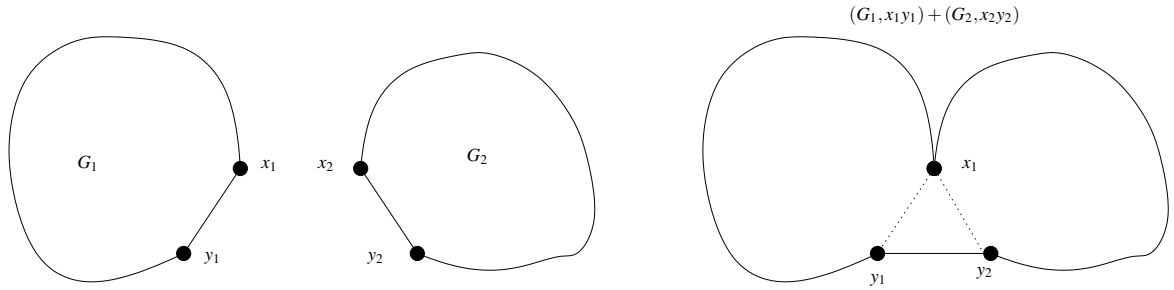


Figure 8.4: Hajós sum

1) Show that $\chi(G) \geq \min\{\chi(G_1), \chi(G_2)\}$.
2) Show that $\chi(G) \geq \max\{\chi(G_1), \chi(G_2)\} - 1$. Give an example for which $\chi(G) = \max\{\chi(G_1), \chi(G_2)\} - 1$.

Exercise 8.10. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2$ two graphs such that $V_1 \cap V_2 = \emptyset$.
The *disjoint union* of G_1 and G_2 is the graph $G_1 + G_2$, defined by $V(G_1 + G_2) = V_1 \cup V_2$ and $E(G_1 + G_2) = E_1 \cup E_2$.
The *join* of G_1 and G_2 is the graph $G_1 \oplus G_2$ defined by $V(G_1 \oplus G_2) = V_1 \cup V_2$ and $E(G_1 \oplus G_2) = E_1 \cup E_2 \cup \{v_1v_2 \mid v_1 \in V_1 \text{ and } v_2 \in V_2\}$.
1) Show that $\chi(G_1 + G_2) = \max\{\chi(G_1), \chi(G_2)\}$ and $\chi(G_1 \oplus G_2) = \chi(G_1) + \chi(G_2)$.
2) The class of *cographs* is defined inductively as follows:

- the graph with one vertex K_1 is a cograph;
- the disjoint union of two cographs is a cograph;
- the join of two cographs is a cograph.

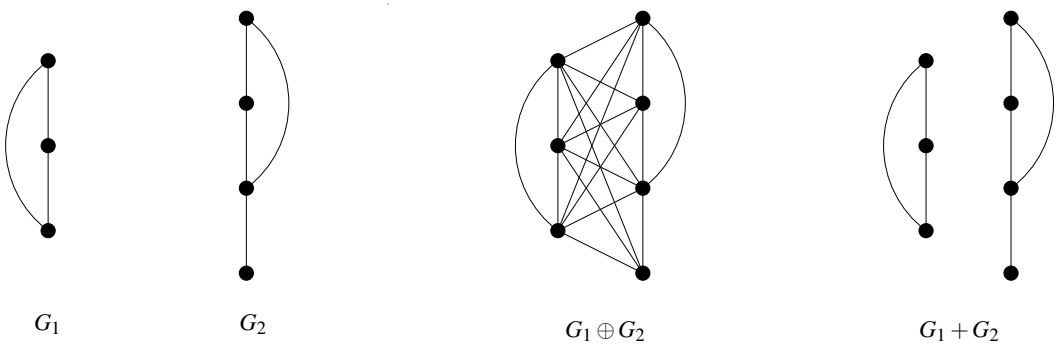


Figure 8.5: Two graphs, their join and their disjoint union

Prove that if G is a cograph then $\chi(G) = \omega(G)$.

Exercise 8.11. Show that, in every k-chromatic graph, there are at least k vertices of degree at least $k-1$.

Exercise 8.12. Show that $\chi'(K_{2n-1}) = \chi'(K_{2n}) = 2n-1$.

Exercise 8.13. Let G be an r-regular bipartite graph and E_0 a set of $r-1$ edges. Show that $G \setminus E_0$ has a perfect matching.

Exercise 8.14. The *cartesian product* of two graphs G and H is the graph $G \times H$ defined by

$$\begin{aligned} V(G\Box H) &= V(G) \times V(H) \\ E(G\Box H) &= \{(a,x)(b,y) \mid a = b \text{ and } xy \in E(H) \text{ or } ab \in E(G) \text{ and } x = y\}. \end{aligned}$$

(a) Show that $\chi'(G\Box K_2) = \Delta(G\Box K_2)$.

(b) Deduce that if H is non-empty (it has at least one edge) and $\chi'(H) = \Delta(H)$ then $\chi'(G\Box H) = \Delta(G\Box H)$.

Bibliography

[1] K. Appel and W. Haken. Every planar map is four colourable. I. Discharging. *Illinois J. Math.* 21, 429–490, 1977.

[2] K. Appel, W. Haken, and J. Koch. Every planar map is four colourable. II. Reducibility. *Illinois J. Math.* 21:491–567, 1977.

[3] K. Appel and W. Haken. Every Planar Map is Four Colourable. *Contemporary Mathematics* 98. American Mathematical Society, Providence, RI, 1989.

[4] A. Beutelspacher and P.-R. Hering. Minimal graphs for which the chromatic number equals the maximal degree. *Ars Combin.* 18:201–216, 1984.

[5] O. V. Borodin and A. V. Kostochka. On an upper bound of a graph's chromatic number, depending on the graph's degree and density. *J. Combin. Theory Ser. B* 23(2-3):247–250, 1977.

[6] R. L. Brooks, On colouring the nodes of a network. *Proc. Cambridge Phil. Soc.* 37:194–197, 1941.

[7] B. Descartes. A three colour problem. *Eureka* 21, 1947.

[8] T. Emden-Weinert, S. Hougardy, and B. Kreuter. Uniquely colourable graphs and the hardness of colouring graphs of large girth. *Combin. Probab. Comput.* 7(4):375–386, 1998.

[9] P. Erdős and R. J. Wilson. On the chromatic index of almost all graphs. *J. Combin. Theory Ser. B* 23:255–257, 1977.

[10] B. Farzad, M. Molloy and B. Reed. $(\Delta - k)$-critical graphs. *J. Combin. Theory Ser. B* 93:173–185, 2005.

[11] M. R. Garey and D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[12] R. P. Gupta. The chromatic index and the degree of a graph. *Not. Amer. Math. Soc.* 13:719, 1966.

[13] I. Holyer. The NP-completeness of edge-coloring. *SIAM J. Computing* 2:225–231, 1981.

[14] A. Johansson, Asymptotic choice number for triangle free graphs. *DIMACS Technical Report* 91-5.

[15] J. Kelly and L. Kelly. Path and circuits in critical graphs. *Amer. J. Math.* 76:786–792, 1954.

[16] D. König. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. *Math. Ann.* 77:453–465, 1916.

[17] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. 25th ACM Symposium on Theory of Computing*, pages 286-293, 1993.

[18] M. Molloy and B. Reed. Colouring graphs when the number of colours is nearly the maximum degree. In *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, pages 462–470 (electronic), New York, 2001. ACM.

[19] J. Mycielski, Sur le coloriage des graphes. *Information Processing Letters* 108(6):412-417, 2008.

[20] B. Reed. ω, Δ, and χ. *J. Graph Theory* 27(4):177–212, 1998.

[21] B. Reed. A strengthening of Brooks' theorem. *J. Combin. Theory Ser. B* 76(2):136–149, 1999.

[22] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas. A new proof of the four colour theorem. *Electron. Res. Announc. Amer. Math. Soc.* 2:17–25, 1996.

[23] N. Robertson, D. P. Sanders, P. D. Seymour and R. Thomas. The four colour theorem. *J. Combin. Theory Ser. B.* 70:2–44, 1997.

[24] D. P. Sanders and Y. Zhao. Planar graphs of maximum degree seven are class I. *J. Combin. Theory Ser. B* 83(2):201–212, 2001.

[25] V. G. Vizing. On an estimate of the chromatic class of a p-graph. *Metody Diskret. Analyz.* 3:25–30, 1964.

[26] V. G. Vizing. Critical graphs with given chromatic index. *Metody Diskret. Analiz* 5:9–17, 1965. [In Russian]

[27] A. A. Zykov. On some properties of linear complexes. *Mat. Sbornik* 24:313–319, 1949. (In Russian).

Chapter 9

Linear programming

The nature of the programmes a computer scientist has to conceive often requires some knowledge in a specific domain of application, for example corporate management, network protocols, sound and video for multimedia streaming,... Linear programming is one of the necessary knowledges to handle optimization problems. These problems come from varied domains as production management, economics, transportation network planning, ... For example, one can mention the composition of train wagons, the electricity production, or the flight planning by airplane companies.

Most of these optimization problems do not admit an optimal solution that can be computed in a reasonable time, that is in polynomial time (See Chapter 3). However, we know how to efficiently solve some particular problems and to provide an optimal solution (or at least quantify the difference between the provided solution and the optimal value) by using techniques from linear programming.

In fact, in 1947, G.B. Dantzig conceived the Simplex Method to solve military planning problems asked by the US Air Force that were written as a linear programme, that is a system of linear equations. In this course, we introduce the basic concepts of linear programming. We then present the Simplex Method, following the book of V. Chvátal [2]. If you want to read more about linear programming, some good references are [6, 1].

The objective is to show the reader how to model a problem with a linear programme when it is possible, to present him different methods used to solve it or at least provide a good approximation of the solution. To this end, we present the *theory of duality* which provide ways of finding good bounds on specific solutions.

We also discuss the practical side of linear programming: there exist very efficient tools to solve linear programmes, e.g. CPLEX [3] and GLPK [4]. We present the different steps leading to the solution of a practical problem expressed as a linear programme.

9.1 Introduction

A *linear programme* is a problem consisting in maximizing or minimizing a linear function while satisfying a finite set of linear constraints.

Linear programmes can be written under the *standard form*:

$$\begin{array}{lrcll} \text{Maximize} & \sum_{j=1}^{n} c_j x_j & & & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for all } 1 \leq i \leq m \\ & x_j & \geq & 0 & \text{for all } 1 \leq j \leq n. \end{array} \tag{9.1}$$

All constraints are inequalities (and not equations) and all variables are non-negative. The variables x_j are referred to as *decision variables*. The function that has to be maximized is called the problem *objective function*.

Observe that a constraint of the form $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$ may be rewritten as $\sum_{j=1}^{n} (-a_{ij}) x_j \leq -b_i$. Similarly, a minimization problem may be transformed into a maximization problem: minimizing $\sum_{j=1}^{n} c_j x_j$ is equivalent to maximizing $\sum_{j=1}^{n} (-c_j) x_j$. Hence, every maximization or minimization problem subject to linear constraints can be reformulated in the standard form (See Exercices 9.1 and 9.2.).

A n-tuple $(x_1, \ldots, x_n)$ satisfying the constraints of a linear programme is a *feasible solution* of this problem. A solution that maximizes the objective function of the problem is called an *optimal solution*. Beware that a linear programme does not necessarily admits a unique optimal solution. Some problems have several optimal solutions while others have none. The later case may occur for two opposite reasons: either there exist no feasible solutions, or, in a sense, there are too many. The first case is illustrated by the following problem.

$$\begin{array}{lrcrcr} \text{Maximize} & 3x_1 & - & x_2 & & \\ \text{Subject to:} & x_1 & + & x_2 & \leq & 2 \\ & -2x_1 & - & 2x_2 & \leq & -10 \\ & & & x_1, x_2 & \geq & 0 \end{array} \tag{9.2}$$

which has no feasible solution (See Exercise 9.3). Problems of this kind are referred to as *unfeasible*. At the opposite, the problem

$$\begin{array}{lrcrcr} \text{Maximize} & x_1 & - & x_2 & & \\ \text{Subject to:} & -2x_1 & + & x_2 & \leq & -1 \\ & -x_1 & - & 2x_2 & \leq & -2 \\ & & & x_1, x_2 & \geq & 0 \end{array} \tag{9.3}$$

has feasible solutions. But none of them is optimal (See Exercise 9.3). As a matter of fact, for every number M, there exists a feasible solution x_1, x_2 such that $x_1 - x_2 > M$. The problems verifying this property are referred to as *unbounded*. Every linear programme satisfies exactly one the following assertions: either it admits an optimal solution, or it is unfeasible, or it is unbounded.

Geometric interpretation.

The set of points in $\mathbb{R}^n$ at which any single constraint holds with equality is a hyperplane in $\mathbb{R}^n$. Thus each constraint is satisfied by the points of a closed half-space of $\mathbb{R}^n$, and the set of feasible solutions is the intersection of all these half-spaces, a convex polyhedron P.

Because the objective function is linear, its level sets are hyperplanes. Thus, if the maximum value of $\mathbf{cx}$ over P is z^*, the hyperplane $\mathbf{cx} = z^*$ is a supporting hyperplane of P. Hence $\mathbf{cx} = z^*$ contains an extreme point (a corner) of P. It follows that the objective function attains its maximum at one of the extreme points of P.

9.2 The Simplex Method

The authors advise you, in a humanist élan, to skip this section if you are not ready to suffer. In this section, we present the principle of the Simplex Method. We consider here only the most general case and voluntarily omit here the degenerate cases to focus only on the basic principle. A more complete presentation can be found for example in [2].

9.2.1 A first example

We illustrate the Simplex Method on the following example:

$$\begin{array}{lrcrcrcr} \text{Maximize} & 5x_1 & + & 4x_2 & + & 3x_3 & & \\ \text{Subject to:} & & & & & & & \\ & 2x_1 & + & 3x_2 & + & x_3 & \leq & 5 \\ & 4x_1 & + & x_2 & + & 2x_3 & \leq & 11 \\ & 3x_1 & + & 4x_2 & + & 2x_3 & \leq & 8 \\ & & & & & x_1,x_2,x_3 & \geq & 0. \end{array} \tag{9.4}$$

The first step of the Simplex Method is to introduce new variables called *slack variables*. To justify this approach, let us look at the first constraint,

$$2x_1 + 3x_2 + x_3 \leq 5. \tag{9.5}$$

For all feasible solution x_1,x_2,x_3, the value of the left member of (9.5) is at most the value of the right member. But, there often is a gap between these two values. We note this gap x_4. In other words, we define $x_4 = 5 - 2x_1 - 3x_2 - x_3$. With this notation, Equation (9.5) can now be written as $x_4 \geq 0$. Similarly, we introduce the variables x_5 and x_6 for the two other constraints of Problem (9.4). Finally, we use the classic notation z for the objective function $5x_1 + 4x_2 + 3x_3$. To summarize, for all choices of x_1,x_2,x_3 we define x_4,x_5,x_6 and z by the formulas

$$\begin{array}{rcrcrcrcr} x_4 & = & 5 & - & 2x_1 & - & 3x_2 & - & x_3 \\ x_5 & = & 11 & - & 4x_1 & - & x_2 & - & 2x_3 \\ x_6 & = & 8 & - & 3x_1 & - & 4x_2 & - & 2x_3 \\ z & = & & & 5x_1 & + & 4x_2 & + & 3x_3. \end{array} \tag{9.6}$$

With these notations, the problem can be written as:

$$\text{Maximize } z \text{ subject to } x_1,x_2,x_3,x_4,x_5,x_6 \geq 0. \tag{9.7}$$

The new variables that were introduced are referred as *slack variables*, when the initial variables are usually called the *decision variables*. It is important to note that Equation (9.6) define an equivalence between (9.4) and (9.7). More precisely:

- Any feasible solution (x_1,x_2,x_3) of (9.4) can be uniquely extended by (9.6) into a feasible solution $(x_1,x_2,x_3,x_4,x_5,x_6)$ of (9.7).

- Any feasible solution $(x_1,x_2,x_3,x_4,x_5,x_6)$ of (9.7) can be reduced by a simple removal of the slack variables into a feasible solution (x_1,x_2,x_3) of (9.4).
- This relationship between the feasible solutions of (9.4) and the feasible solutions of (9.7) allows to produce the optimal solution of (9.4) from the optimal solutions of (9.7) and *vice versa*.

The Simplex strategy consists in finding the optimal solution (if it exists) by successive improvements. If we have found a feasible solution (x_1,x_2,x_3) of (9.7), then we try to find a new solution $(\bar{x_1},\bar{x_2},\bar{x_3})$ which is better in the sense of the objective function:

$$5\bar{x_1}+4\bar{x_2}+3\bar{x_3} \geq 5x_1+4x_2+3x_3.$$

By repeating this process, we obtain at the end an optimal solution.

To start, we first need a feasible solution. To find one in our example, it is enough to set the decision variables x_1,x_2,x_3 to zero and to evaluate the slack variables x_4,x_5,x_6 using (9.6). Hence, our initial solution,

$$x_1=0,x_2=0,x_3=0,x_4=5,x_5=11,x_6=8 \tag{9.8}$$

gives the result $z=0$.

We now have to look for a new feasible solution which gives a larger value for z. Finding such a solution is not hard. For example, if we keep $x_2=x_3=0$ and increase the value of x_1, then we obtain $z=5x_1\geq 0$. Hence, if we keep $x_2=x_3=0$ and if we set $x_1=1$, then we obtain $z=5$ (and $x_4=3,x_5=7,x_6=5$). A better solution is to keep $x_2=x_3=0$ and to set $x_1=2$; we then obtain $z=10$ (and $x_4=1,x_5=3,x_6=2$). However, if we keep $x_2=x_3=0$ and if we set $x_1=3$, then $z=15$ and $x_4=x_5=x_6=-1$, breaking the constraint $x_i\geq 0$ for all i. The conclusion is that one can not increase x_1 as much as one wants. The question then is: how much can x_1 be raised (when keeping $x_2=x_3=0$) while satisfying the constraints $(x_4,x_5,x_6\geq 0)$?

The condition $x_4=5-2x_1-3x_2-x_3\geq 0$ implies $x_1\leq\frac{5}{2}$. Similarly, $x_5\geq 0$ implies $x_1\leq\frac{11}{4}$ and $x_6\geq 0$ implies $x_1\leq\frac{8}{3}$. The first bound is the strongest one. Increasing x_1 to this bound gives the solution of the next step:

$$x_1=\frac{5}{2},x_2=0,x_3=0,x_4=0,x_5=1,x_6=\frac{1}{2} \tag{9.9}$$

which gives a result $z=\frac{25}{2}$ improving the last value $z=0$ of (9.8).

Now, we have to find a new feasible solution that is better than (9.9). However, this task is not as simple as before. Why? As a matter of fact, we had at disposal the feasible solution of (9.8), but also the system of linear equations (9.6) which led us to a better feasible solution. Thus, we should build a new system of linear equations related to (9.9) in the same way as (9.6) is related to (9.8).

Which properties should have this new system? Note first that (9.6) express the strictly positive variables of (9.8) in function of the null variables. Similarly, the new system has to express the strictly positive variables of (9.9) in function of the null variables of (9.9): x_1,x_5,x_6 (and z) in function of x_2,x_3 and x_4. In particular, the variable x_1, whose value just increased

from zero to a strictly positive value, has to go to the left side of the new system. The variable x_4, which is now null, has to take the opposite move.

To build this new system, we start by putting x_1 on the left side. Using the first equation of (9.6), we write x_1 in function of x_2, x_3, x_4:

$$x_1 = \frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4 \tag{9.10}$$

Then, we express x_5, x_6 and z in function of x_2, x_3, x_4 by substituting the expression of x_1 given by (9.10) in the corresponding lines of (9.6).

$$\begin{aligned}
x_5 &= 11 - 4\left(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4\right) - x_2 - 2x_3 \\
&= 1 + 5x_2 + 2x_4, \\
x_6 &= 8 - 3\left(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4\right) - 4x_2 - 2x_3 \\
&= \frac{1}{2} + \frac{1}{2}x_2 - \frac{1}{2}x_3 + \frac{3}{2}x_4, \\
z &= 5\left(\frac{5}{2} - \frac{3}{2}x_2 - \frac{1}{2}x_3 - \frac{1}{2}x_4\right) + 4x_2 + 3x_3 \\
&= \frac{25}{2} - \frac{7}{2}x_2 + \frac{1}{2}x_3 - \frac{5}{2}x_4.
\end{aligned}$$

So the new system is

$$\begin{array}{rcrcrcrcrl}
x_1 &=& \frac{5}{2} &-& \frac{3}{2} & x_2 &-& \frac{1}{2} & x_3 &-& \frac{1}{2} & x_4 \\
x_5 &=& 1 &+& 5 & x_2 & & & & + & 2 & x_4 \\
x_6 &=& \frac{1}{2} &+& \frac{1}{2} & x_2 &-& \frac{1}{2} & x_3 &+& \frac{3}{2} & x_4 \\
z &=& \frac{25}{2} &-& \frac{7}{2} & x_2 &+& \frac{1}{2} & x_3 &-& \frac{5}{2} & x_4.
\end{array} \tag{9.11}$$

As done at the first iteration, we now try to increase the value of z by increasing a right variable of the new system, while keeping the other right variables at zero. Note that raising x_2 or x_4 would lower the value of z, against our objective. So we try to increase x_3. How much? The answer is given by (9.11) : with $x_2 = x_4 = 0$, the constraint $x_1 \geq 0$ implies $x_3 \leq 5$, $x_5 \geq 0$ impose no restriction and $x_6 \geq 0$ implies that $x_3 \leq 1$. To conclude $x_3 = 1$ is the best we can do, and the new solution is

$$x_1 = 2, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1, x_6 = 0 \tag{9.12}$$

and the value of z increases from 12.5 to 13. As stated, we try to obtain a better solution but also a system of linear equations associated to (9.12). In this new system, the (strictly) positive variables x_2, x_4, x_6 have to appear on the right. To build this new system, we start by handling the new left variable, x_3. Thanks to the third equation of (9.11) we rewrite x_3 and by substitution

in the remaining equations of (9.11) we obtain:

$$\begin{array}{rcrcrcrcr} x_3 & = & 1 & + & x_2 & + & 3x_4 & - & 2x_6 \\ x_1 & = & 2 & - & 2x_2 & - & 2x_4 & + & x_6 \\ x_5 & = & 1 & + & 5x_2 & + & 2x_4 & & \\ z & = & 13 & - & 3x_2 & - & x_4 & - & x_6. \end{array} \tag{9.13}$$

It is now time to do the third iteration. First, we have to find a variable of the right side of (9.13) whose increase would result in an increase of the objective z. But there is no such variable, as any increase of x_2, x_4 or x_6 would lower z. We are stuck. In fact, this deadlock indicates that the last solution is optimal. Why? The answer lies in the last line of (9.13):

$$z = 13 - 3x_2 - x_4 - x_6. \tag{9.14}$$

The last solution (9.12) gives a value $z = 13$; proving that this solution is optimal boils down to prove that any feasible solution satisfies $z \leq 13$. As any feasible solution $x_1, x_2, \ldots, x_6$ satisfies the inequalities $x_2 \geq 0, x_4 \geq 0, x_6 \geq 0$, then $z \leq 13$ directly derives from (9.14).

9.2.2 The dictionaries

More generally, given a problem

$$\begin{array}{lrcll} \text{Maximize} & \sum_{j=1}^{n} c_j x_j & & & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij} x_j & \leq & b_i & \text{for all } 1 \leq i \leq m \\ & x_j & \geq & 0 & \text{for all } 1 \leq j \leq n \end{array} \tag{9.15}$$

we first introduce the *slack variables* $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ and we note the objective function z. That is, we define

$$\begin{array}{rcll} x_{n+i} & = & b_i - \sum_{j=1}^{n} a_{ij} x_j & \text{for all } 1 \leq i \leq m \\ z & = & \sum_{j=1}^{n} c_j x_j & \end{array} \tag{9.16}$$

In the framework of the Simplex Method, each feasible solution $(x_1, x_2, \ldots, x_n)$ of (9.15) is represented by $n+m$ positive or null numbers $x_1, x_2, \ldots, x_{n+m}$, with $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ defined by (9.16). At each iteration, the Simplex Method goes from one feasible solution $(x_1, x_2, \ldots, x_{n+m})$ to an other feasible solution $(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{n+m})$, which is better in the sense that

$$\sum_{j=1}^{n} c_j \bar{x}_j > \sum_{j=1}^{n} c_j x_j.$$

As we have seen in the example, it is convenient to associate a system of linear equations to each feasible solution. As a matter of fact, it allows to find better solutions in an easy way. The technique is to translate the choices of the values of the variables of the right side of the system into the variables of the left side and in the objective function as well. These systems have been named *dictionaries* by J.E. Strum (1972). Thus, every dictionary associated to (9.15) is a system of equations whose variables $x_{n+1}, x_{n+2}, \ldots, x_{n+m}$ and z are expressed in function of $x_1, x_2, \ldots, x_n$. These $n+m+1$ variables are closely linked and every dictionary express these dependencies.

Property 9.1. *Any feasible solution of the equations of a dictionary is also a feasible solution of (9.16) and* vice versa*.*

For example, for any choice of $x_1, x_2, \ldots, x_6$ and of z, the three following assertions are equivalent:

- $(x_1, x_2, \ldots, x_6, z)$ is a feasible solution of (9.6);
- $(x_1, x_2, \ldots, x_6, z)$ is a feasible solution of (9.11);
- $(x_1, x_2, \ldots, x_6, z)$ is a feasible solution of (9.13).

From this point of view, the three dictionaries (9.6), (9.11) and (9.13) contain the same information on the dependencies between the seven variables. However, each dictionary present this information in a specific way. (9.6) suggests that the values of the variables x_1, x_2 and x_3 can be chosen at will while the values of x_4, x_5, x_6 and z are fixed. In this dictionary, the decision variables x_1, x_2, x_3 act as independent variables while the slack variables x_4, x_5, x_6 are related to each other. In the dictionary (9.13), the independent variables are x_2, x_4, x_6 and the related ones are x_3, x_1, x_5, z.

Property 9.2. *The equations of a dictionary have to express m variables among $x_1, x_2, \ldots, x_{n+m}, z$ in function of the n remaining others.*

Properties 9.1 and 9.2 define what a dictionary is. In addition to these two properties, the dictionaries (9.6),(9.11) and (9.13) have the following property.

Property 9.3. *When putting the right variables to zero, one obtains a feasible solution by evaluating the left variables.*

The dictionaries that have this last property are called *feasible dictionaries*. As a matter of fact, any feasible dictionary describes a feasible solution. However, all feasible solutions cannot be described by a feasible dictionary. For example, no dictionary describe the feasible solution $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 2$, $x_5 = 5$, $x_6 = 3$ of (9.4). The feasible solutions that can be described by dictionaries are referred as *basic solutions*. The Simplex Method explores only basic solutions and ignores all other ones. But this is valid because if an optimal solution exists, then there is an optimal and basic solution. Indeed, if a feasible solution cannot be improved by the Simplex Method, then increasing any of the n right variables to a positive value never increases the objective function. In such case, the objective function must be written as a linear function of these variables in which all the coefficient are non-positive, and thus the objective function is clearly maximum when all the right variables equal zero. For example, it was the case in (9.14).

9.2.3 Finding an initial solution

In the previous examples, the initialization of the Simplex Method was not a problem. As a matter of fact, we carefully chose problems with all b_i non-negative. This way $x_1 = 0$, $x_2 = 0$,

$\cdots, x_n = 0$ was a feasible solution and the dictionary was easily built. These problems are called *problems with a feasible origin*.

What happens when confronted with a problem with an unfeasible origin? Two difficulties arise. First, a feasible solution can be hard to find. Second, even if we find a feasible solution, a feasible dictionary has then to be built. A way to solve these difficulties is to use another problem called *auxiliary problem*:

$$\begin{array}{lrcll} \text{Minimise} & x_0 & & & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij}x_j - x_0 & \leq & b_i & (i = 1,2,\cdots,m) \\ & x_j & \geq & 0 & (j = 0,1,\cdots,n). \end{array}$$

A feasible solution of the auxiliary problem is easily available: it is enough to set $x_j = 0 \forall j \in [1 \ldots n]$ and to give to x_0 a big enough value. It is now easy to see that the original problem has a feasible solution if and only if the auxiliary problem has a feasible solution with $x_0 = 0$. In other words, the original problem has a feasible solution if the optimal value of the auxiliary problem is null. Thus, the idea is to first solve the auxiliary problem. Let see the details on an example.

$$\begin{array}{lrcrcrcr} \text{Maximise} & x_1 & - & x_2 & + & x_3 & & \\ \text{Subject to :} & & & & & & & \\ & 2x_1 & - & x_2 & + & 2x_3 & \leq & 4 \\ & 2x_1 & - & 3x_2 & + & x_3 & \leq & -5 \\ & -x_1 & + & x_2 & - & 2x_3 & \leq & -1 \\ & & & & & x_1,x_2,x_3 & \geq & 0 \end{array}$$

$$\begin{array}{lrcrcrcrcr} \text{Maximise} & -x_0 & & & & & & & & \\ \text{Subject to:} & & & & & & & & & \\ & 2x_1 & - & x_2 & + & 2x_3 & - & x_0 & \leq & 4 \\ & 2x_1 & - & 3x_2 & + & x_3 & - & x_0 & \leq & -5 \\ & -x_1 & + & x_2 & - & 2x_3 & - & x_0 & \leq & -1 \\ & & & & & x_1,x_2,x_3,x_0 & \geq & 0 & & \end{array}$$

We introduce the slack variables. We obtain the dictionary:

$$\begin{array}{rcrcrcrcrcr} x_4 & = & 4 & - & 2x_1 & + & x_2 & - & 2x_3 & + & x_0 \\ x_5 & = & -5 & - & 2x_1 & + & 3x_2 & - & x_3 & + & x_0 \\ x_6 & = & -1 & + & x_1 & - & x_2 & + & 2x_3 & + & x_0 \\ w & = & & & & & & & & - & x_0. \end{array} \tag{9.17}$$

Note that this dictionary is not feasible. However it can be transformed into a feasible one by operating a simple pivot , x_0 entering the basis as x_5 exits it:

$$\begin{array}{rcrcrcrcrcr} x_0 & = & 5 & + & 2x_1 & - & 3x_2 & + & x_3 & + & x_5 \\ x_4 & = & 9 & & & - & 2x_2 & - & x_3 & + & x_5 \\ x_6 & = & 4 & +3 & x_1 & - & 4x_2 & + & 3x_3 & + & x_5 \\ \hline w & = & -5 & - & 2x_1 & + & 3x_2 & - & x_3 & - & x_5. \end{array}$$

More generally, the auxiliary problem can be written as

$$\begin{array}{lrcll} \text{Maximise} & -x_0 & & & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij}x_j - x_0 & \leq & b_i & (i=1,2,\cdots,m) \\ & x_j & \geq & 0 & (j=0,1,2,\cdots,n) \end{array}$$

and the associated dictionary is

$$\begin{array}{rcrcrcl} x_{n+i} & = & b_i - & \sum_{j=1}^{n} a_{ij}x_j & + & x_0 & (i=1,2,\cdots,m) \\ w & = & & & - & x_0 & \end{array}$$

This dictionary can be made feasible by pivoting x_0 with the variable the "most unfeasible", that is the exiting variable x_{n+k} is the one with $b_k \leq b_i$ for all i. After the pivot, the variable x_0 has value $-b_k$ and each x_{n+i} has value $b_i - b_k$. All these values are non negative. We are now able to solve the auxiliary problem using the simplex method. Let us go back to our example.

After the first iteration with x_2 entering and x_6 exiting, we get:

$$\begin{array}{rcrcrcrcrcr} x_2 & = & 1 & + & 0.75x_1 & + & 0.75x_3 & + & 0.25x_5 & - & 0.25x_6 \\ x_0 & = & 2 & - & 0.25x_1 & - & 1.25x_3 & + & 0.25x_5 & + & 0.75x_6 \\ x_4 & = & 7 & - & 1.5x_1 & - & 2.5x_3 & + & 0.5x_5 & + & 0.5x_6 \\ \hline w & = & -2 & + & 0.25x_1 & + & 1.25x_3 & - & 0.25x_5 & - & 0.75x_6. \end{array}$$

After the second iteration with x_3 entering and x_0 exiting:

$$\begin{array}{rcrcrcrcrcr} x_3 & = & 1.6 & - & 0.2x_1 & + & 0.2x_5 & + & 0.6x_6 & - & 0.8x_0 \\ x_2 & = & 2.2 & + & 0.6x_1 & + & 0.4x_5 & + & 0.2x_6 & - & 0.6x_0 \\ x_4 & = & 3 & - & x_1 & & & - & x_6 & + & 2x_0 \\ \hline w & = & & & & & & & & - & x_0. \end{array} \tag{9.18}$$

The last dictionary (9.18) is optimal. As the optimal value of the auxiliary problem is null, this dictionary provides a feasible solution of the original problem: $x_1 = 0, x_2 = 2.2, x_3 = 1.6$. Moreover, (9.18) can be easily transformed into a feasible dictionary of the original problem. To obtain the first three lines of the desired dictionary, it is enough to copy the first three lines while removing the terms with x_0:

$$\begin{array}{rcrcrcrcr} x_3 & = & 1.6 & - & 0.2x_1 & + & 0.2x_5 & + & 0.6x_6 \\ x_2 & = & 2.2 & + & 0.6x_1 & + & 0.4x_5 & + & 0.2x_6 \\ x_4 & = & 3 & - & x_1 & & & - & x_6 \end{array} \tag{9.19}$$

To obtain the last line, we express the original objective function

$$z = x_1 - x_2 + x_3 \tag{9.20}$$

in function of the variables outside the basis x_1, x_5, x_6. To do so, we replace the variables of (9.20) by (9.19) and we get:

$$\begin{aligned} z = x_1 - (2.2 + 0.6x_1 + 0.4x_5 + 0.2x_6) + (1.6 - 0.2x_1 + 0.2x_5 + 0.6x_6) \\ z = -0.6 + 0.2x_1 - 0.2x_5 + 0.4x_6 \end{aligned} \tag{9.21}$$

The desired dictionary then is:

$$\begin{array}{rcrcrcrcr} x_3 & = & 1.6 & - & 0.2x_1 & + & 0.2x_5 & + & 0.6x_6 \\ x_2 & = & 2.2 & + & 0.6x_1 & + & 0.4x_5 & + & 0.2x_6 \\ x_4 & = & 3 & - & x_1 & & & - & x_6 \\ \hline z & = & -0.6 & + & 0.2x_1 & - & 0.2x_5 & + & 0.4x_6 \end{array}$$

This strategy is known as the *Simplex Method in two phases*. During the first phase, we set and solve the auxiliary problem. If the optimal value is null, we do the second phase consisting in solving the original problem. Otherwise, the original problem is not feasible.

9.3 Duality of linear programming

Any maximization linear programme has a corresponding minimization problem called the *dual problem*. Any feasible solution of the dual problem gives an upper bound on the optimal value of the initial problem, which is called the *primal*. Reciprocally, any feasible solution of the primal provides a lower bound on the optimal value of the dual problem. Actually, if one of both problems admits an optimal solution, then the other problem does as well and the optimal solutions match each other. This section is devoted to this result also known as the *Duality Theorem*. Another interesting application of the dual problem is that, in some problems, the variables of the dual have some useful interpretation.

9.3.1 Motivations: providing upper bounds on the optimal value

A way to quickly estimate the optimal value of a maximization linear programme simply consists in computing a feasible solution whose value is sufficiently large. For instance, let us consider the following problem formulated in Problem 9.4. The solution $(0,0,1,0)$ gives us a lower bound of 5 for the optimal value z^*. Even better, we get $z^* \geq 22$ by considering the solution $(3,0,2,0)$. Of course, doing so, we have no way to know how close to the optimal value the computed lower bound is.

Problem 9.4.

$$\begin{array}{rcrcl} \text{Maximize} & & 4x_1 + x_2 + 5x_3 + 3x_4 & & \\ \text{Subject to:} & & x_1 - x_2 - x_3 + 3x_4 & \leq & 1 \\ & & 5x_1 + x_2 + 3x_3 + 8x_4 & \leq & 55 \\ & & -x_1 + 2x_2 + 3x_3 - 5x_4 & \leq & 3 \\ & & x_1, x_2, x_3, x_4 & \geq & 0 \end{array}$$

The previous approach provides lower bounds on the optimal value. However, this intuitive method is obviously less efficient than the Simplex Method and this approach provides no clue about the optimality (or not) of the obtained solution. To do so, it is interesting to have upper bounds on the optimal value. This is the main topic of this section.

How to get an upper bound for the optimal value in the previous example? A possible approach is to consider the constraints. For instance, multiplying the second constraint by $\frac{5}{3}$, we get that $z^* \leq \frac{275}{3}$. Indeed, for any $x_1, x_2, x_3, x_4 \geq 0$:

$$\begin{aligned} 4x_1 + x_2 + 5x_3 + 3x_4 &\leq \frac{25}{3}x_1 + \frac{5}{3}x_2 + 5x_3 + \frac{40}{3}x_4 = (5x_1 + x_2 + 3x_3 + 8x_4) \times \frac{5}{3} \\ &\leq 55 \times \frac{5}{3} = \frac{275}{3} \end{aligned}$$

In particular, the above inequality is satisfied by any optimal solution. Therefore, $z^* \leq \frac{275}{3}$. Let us try to improve this bound. For instance, we can add the second constraint to the third one. This gives, for any $x_1, x_2, x_3, x_4 \geq 0$:

$$\begin{aligned} 4x_1 + x_2 + 5x_3 + 3x_4 &\leq 4x_1 + 3x_2 + 6x_3 - 3x_4 \\ &\leq (5x_1 + x_2 + 3x_3 + 8x_4) + (-x_1 + 2x_2 + 3x_3 - 5x_4) \\ &\leq 55 + 3 = 58 \end{aligned}$$

Hence, $z^* \leq 58$.

More formally, we try to upper bound the optimal value by a linear combination of the constraints. Precisely, for all i, let us multiply the i^{th} constraint by $y_i \geq 0$ and then sum the resulting constraints. In the previous two examples, we had $(y_1, y_2, y_3) = (0, \frac{5}{3}, 0)$ and $(y_1, y_2, y_3) = (0, 1, 1)$. More generally, we obtain the following inequality:

$$\begin{aligned} & y_1(x_1 - x_2 - x_3 + 3x_4) + y_2(5x_1 + x_2 + 3x_3 + 8x_4) + y_3(-x_1 + 2x_2 + 3x_3 - 5x_4) \\ = \; & (y_1 - 5y_2 - y_3)x_1 + (-y_1 + y_2 + 2y_3)x_2 + (-y_1 + 3y_2 + 3y_3)x_3 + (3y_1 + 8y_2 - 5y_3)x_4 \\ \leq \; & y_1 + 55y_2 + 3y_3 \end{aligned}$$

For this inequality to provide an upper bound of $4x_1 + x_2 + 5x_3 + 3x_4$, we need to ensure that, for all $x_1, x_2, x_3, x_4 \geq 0$,

$$\begin{aligned} & 4x_1 + x_2 + 5x_3 + 3x_4 \\ \leq \; & (y_1 - 5y_2 - y_3)x_1 + (-y_1 + y_2 + 2y_3)x_2 + (-y_1 + 3y_2 + 3y_3)x_3 + (3y_1 + 8y_2 - 5y_3)x_4. \end{aligned}$$

That is, $y_1 - 5y_2 - y_3 \geq 4$, $-y_1 + y_2 + 2y_3 \geq 1$, $-y_1 + 3y_2 + 3y_3 \geq 5$, and $3y_1 + 8y_2 - 5y_3 \geq 3$.

Combining all inequalities, we obtain the following minimization linear programme:

$$\begin{aligned} \text{Minimize} \quad & y_1 + 55y_2 + 3y_3 \\ \text{Subject to:} \quad & \\ & y_1 - 5y_2 - y_3 \geq 4 \\ & -y_1 + y_2 + 2y_3 \geq 1 \\ & -y_1 + 3y_2 + 3y_3 \geq 5 \\ & 3y_1 + 8y_2 - 5y_3 \geq 3 \\ & y_1, y_2, y_3 \geq 0 \end{aligned}$$

This problem is called the *dual* of the initial maximization problem.

9.3.2 Dual problem

We generalize the example given in Subsection 9.3.1. Consider the following general maximization linear programme:

Problem 9.5.

$$\begin{array}{rll} \text{Maximize} & \sum_{j=1}^{n} c_j x_j & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i & \text{for all } 1 \leq i \leq m \\ & x_j \geq 0 & \text{for all } 1 \leq j \leq n \end{array}$$

Problem 9.5 is called the *primal*. The matricial formulation of this problem is

$$\begin{array}{rl} \text{Maximize} & \mathbf{c}^T \mathbf{x} \\ \text{Subject to:} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & x \geq \mathbf{0} \end{array}$$

where $\mathbf{x}^T = [x_1, \ldots, x_n]$ and $\mathbf{c}^T = [c_1, \ldots, c_n]$ are vectors in $\mathbb{R}^n$, and $\mathbf{b}^T = [b_1, \ldots, b_m] \in \mathbb{R}^m$, and $\mathbf{A} = [a_{ij}]$ is a matrix in $\mathbb{R}^{m \times n}$.

To find an upper bound on $\mathbf{c}^T\mathbf{x}$, we aim at finding a vector $\mathbf{y}^T = [y_1, \ldots, y_m] \geq 0$ such that, for all feasible solutions $\mathbf{x} \geq 0$ of the initial problem, $\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{A}\mathbf{x} \leq \mathbf{y}^T\mathbf{b} = \mathbf{b}^T\mathbf{y}$, that is:

$$\begin{array}{rl} \text{Minimize} & \mathbf{b}^T \mathbf{y} \\ \text{Subject to:} & \mathbf{A}^T\mathbf{y} \geq \mathbf{c} \\ & \mathbf{y} \geq \mathbf{0} \end{array}$$

In other words, the *dual* of Problem 9.5 is defined by:

Problem 9.6.

$$\begin{array}{rll} \text{Minimize} & \sum_{i=1}^{m} b_i y_i & \\ \text{Subject to:} & \sum_{i=1}^{m} a_{ij} y_i \geq c_j & \text{for all } 1 \leq j \leq n \\ & y_i \geq 0 & \text{for all } 1 \leq i \leq m \end{array}$$

Notice that the dual of a maximization problem is a minimization problem. Moreover, there is a one-to-one correspondence between the m constraints of the primal $\sum_{j=1\ldots n} a_{ij}x_j \leq b_i$ and the m variables y_i of the dual. Similarly, the n constraints $\sum_{i=1}^{m} a_{ij}y_i \geq c_j$ of the dual correspond one-to-one to the n variables x_i of the primal.

Problem 9.6, which is the dual of Problem 9.5, can be equivalently formulated under the standard form as follows.

$$\begin{array}{rll} \text{Maximize} & \sum_{i=1}^{m} (-b_i) y_i & \\ \text{Subject to:} & \sum_{i=}^{m} (-a_{ij}) y_i \leq -c_j & \text{for all } 1 \leq j \leq n \\ & y_i \geq 0 & \text{for all } 1 \leq i \leq m \end{array} \tag{9.22}$$

Then, the dual of Problem 9.22 has the following formulation which is equivalent to Problem 9.5.

$$\begin{array}{rll} \text{Minimize} & \sum_{j=1}^{n} (-c_j) x_j & \\ \text{Subject to:} & \sum_{j=1}^{n} (-a_{ij}) x_j \geq -b_i & \text{for all } 1 \leq i \leq m \\ & x_j \geq 0 & \text{for all } 1 \leq j \leq n \end{array} \tag{9.23}$$

We deduce the following lemma.

Lemma 9.7. *If D is the dual of a problem P, then the dual of D is P. Informally, the dual of the dual is the primal.*

9.3.3 Duality Theorem

An important aspect of duality is that feasible solutions of the primal and the dual are related.

Lemma 9.8. *Any feasible solution of Problem 9.6 yields an upper bound for Problem 9.5. In other words, the value given by any feasible solution of the dual of a problem is an upper bound for the primal problem.*

Proof. Let $(y_1,\dots,y_m)$ be a feasible solution of the dual and $(x_1,\dots,x_n)$ be a feasible solution of the primal. Then,

$$\sum_{j=1}^{n} c_j x_j \leq \sum_{j=}^{n} \left(\sum_{i=}^{m} a_{ij} y_i \right) x_j \leq \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j \right) y_i \leq \sum_{i=1}^{m} b_i y_i.$$

□

Corollary 9.9. *If $(y_1,\dots,y_m)$ is a feasible solution of the dual of a problem (Problem 9.6) and $(x_1,\dots,x_n)$ is a feasible solution of the corresponding primal (Problem 9.5) such that $\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i$, then both solutions are optimal.*

Corollary 9.9 states that if we find two solutions for the dual and the primal achieving the same value, then this is a certificate of the optimality of these solutions. In particular, in that case (if they are feasible), both the primal and the dual problems have same optimal value.

For instance, we can easily verify that $(0,14,0,5)$ is a feasible solution for Problem 9.4 with value 29. On the other hand, $(11,0,6)$ is a feasible solution for the dual with same value. Hence, the optimal solutions for the primal and for the dual coincide and are equal to 29.

In general, it is not immediate that any linear programme may have such certificate of optimality. In other words, for any feasible linear programme, can we find a solution of the primal problem and a solution of the dual problem that achieve the same value (thus, this value would be optimal)? One of the most important result of the linear programming is the duality theorem that states that it is actually always the case: for any feasible linear programme, the primal and the dual problems have the same optimal solution. This theorem has been proved by D. Gale, H.W. Kuhn and A. W. Tucker [5] and comes from discussions between G.B. Dantzig and J. von Neumann during Fall 1947.

Theorem 9.10 (DUALITY THEOREM)**.** *If the primal problem defined by Problem 9.5 admits an optimal solution $(x_1^*,\dots,x_n^*)$, then the dual problem (Problem 9.6) admits an optimal solution $(y_1^*,\dots,y_m^*)$, and*

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

Proof. The proof consists in showing how a feasible solution $(y_1^*, \ldots, y_m^*)$ of the dual can be obtained thanks to the Simplex Method, so that $z^* = \sum_{i=1}^m b_i y_i^*$ is the optimal value of the primal. The result then follows from Lemma 9.8.

Let us assume that the primal problem has been solved by the Simplex Method. For this purpose, the slack variables have been defined by

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \quad \text{for } 1 \leq i \leq m.$$

Moreover, the last line of the last dictionary computed during the Simplex Method gives the optimal value z^* of the primal in the following way: for any feasible solution $(x_1, \ldots, x_n)$ of the primal we have

$$z = \sum_{j=1}^{n} c_j x_j = z^* + \sum_{i=1}^{n+m} \bar{c}_i x_i.$$

Recall that, for all $i \leq n+m$, $\bar{c}_i$ is non-positive, and that it is null if x_i is one of the basis variables. We set

$$y_i^* = -\bar{c}_{n+i} \qquad \text{for } 1 \leq i \leq m.$$

Then, by definition of the y_i^*'s and the x_{n+i}'s for $1 \leq i \leq m$, we have

$$\begin{aligned} z = \sum_{j=1}^{n} c_j x_j &= z^* + \sum_{i=1}^{n} \bar{c}_i x_i - \sum_{i=1}^{m} y_i^* \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right) \\ &= \left(z^* - \sum_{i=1}^{m} y_i^* b_i \right) + \sum_{j=1}^{n} \left(\bar{c}_j + \sum_{i=1}^{m} a_{ij} y_i^* \right) x_j. \end{aligned}$$

Since this equation must be true whatever be the affectation of the x_i's and since the $\bar{c}_i$'s are non-positive, this leads to

$$\begin{aligned} z^* &= \sum_{i=1}^{m} y_i^* b_i \quad \text{and} \\ c_j &= \bar{c}_j + \sum_{i=1}^{m} a_{ij} y_i^* \leq \sum_{i=1}^{m} a_{ij} y_i^* \quad \text{for all } 1 \leq j \leq n. \end{aligned}$$

Hence, $(y_1^*, \ldots, y_m^*)$ defined as above is a feasible solution achieving the optimal value of the primal. By Lemma 9.8, this is an optimal solution of the dual. □

9.3.4 Relation between primal and dual

By the Duality Theorem and Lemma 9.7, a linear programme admits a solution if and only if its dual admits a solution. Moreover, according to Lemma 9.8, if a linear programme is unbounded,

then its dual is not feasible. Reciprocally, if a linear programme admits no feasible solution, then its dual is unbounded. Finally, it is possible that both a linear programme and its dual have no feasible solution as shown by the following example.

$$\begin{array}{rrcl} \text{Maximize} & 2x_1 - x_2 & & \\ \text{Subject to:} & x_1 - x_2 & \leq & 1 \\ & -x_1 + x_2 & \leq & -2 \\ & x_1, x_2 & \geq & 0 \end{array}$$

Besides the fact it provides a certificate of optimality, the Duality Theorem has also a practical interest in the application of the Simplex Method. Indeed, the time-complexity of the Simplex Method mainly yields in the number of constraints of the considered linear programme. Hence, when dealing with a linear programme with few variables and many constraints, it will be more efficient to apply the Simplex Method on its dual.

Another interesting application of the Duality Theorem is that it is possible to compute an optimal solution for the dual problem from an optimal solution of the primal. Doing so gives an easy way to test the optimality of a solution. Indeed, if you have a feasible solution of some linear programme, then a solution of the dual problem can be derived (as explained below). Then the initial solution is optimal if and only if the solution obtained for the dual is feasible and leads to the same value.

More formally, the following theorems can be proved

Theorem 9.11 (Complementary Slackness)**.** *Let $(x_1, \ldots, x_n)$ be a feasible solution of Problem 9.5 and $(y_1, \ldots, y_m)$ be a feasible solution of Problem 9.6. These are optimal solutions if and only if*

$$\sum_{i=1}^{m} a_{ij} y_i = c_j, \quad \text{or } x_j = 0, \quad \text{or both} \quad \text{for all } 1 \leq j \leq n, \text{ and}$$
$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad \text{or } y_i = 0, \quad \text{or both} \quad \text{for all } 1 \leq i \leq m.$$

Proof. First, we note that since x and y are feasible $(b_i - \sum_{j=1}^{n} a_{ij} x_j) y_i \geq 0$ and $(\sum_{i=1}^{m} a_{ij} y_i - c_j) x_j \geq 0$. Summing these inequalities over i and j, we obtain

$$\sum_{i=1}^{m} \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right) y_i \geq 0 \tag{9.24}$$
$$\sum_{j=1}^{n} \left(\sum_{i=1}^{n} a_{ij} y_i - c_j \right) x_j \geq 0 \tag{9.25}$$

Adding Inequalities 9.24 and 9.25 and using the strong duality theorem, we obtain

$$\sum_{i=1}^{m} b_i y_i - \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_j y_i + \sum_{j=1}^{n} \sum_{i=1}^{m} a_{ij} y_i x_j - \sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i - \sum_{j=1}^{n} c_j x_j = 0.$$

Therefore, Inequalities 9.24 and 9.25 must be equalities. As the variables are positive, we further get that

$$\text{for all } i, \quad \left(b_i - \sum_{j=1}^{n} a_{ij}x_j\right) y_i = 0$$
$$\text{and for all } j, \quad \left(\sum_{i=1}^{m} a_{ij}y_i - c_j\right) x_j = 0.$$

A product is equal to zero if one of its two members is null and we obtain the desired result. □

Theorem 9.12. *A feasible solution $(x_1,\ldots,x_n)$ of Problem 9.5 is optimal if and only if there is a feasible solution $(y_1,\ldots,y_m)$ of Problem 9.6 such that:*

$$\begin{array}{rr} \sum_{i=}^{m} a_{ij}y_i = c_j \quad if & x_j > 0 \\ y_i = 0 \quad if & \sum_{j=1}^{m} a_{ij}x_j < b_i \end{array} \tag{9.26}$$

Note that, if Problem 9.5 admits a non-degenerated solution $(x_1,\ldots,x_n)$, i.e., $x_i > 0$ for any $i \leq n$, then the system of equations in Theorem 9.12 admits a unique solution.

Optimality certificates - Examples. Let see how to apply this theorem on two examples.

Let us first examine the statement that

$$x_1^* = 2,\ x_2^* = 4,\ x_3^* = 0,\ x_4^* = 0,\ x_5^* = 7,\ x_6^* = 0$$

is an optimal solution of the problem

$$\begin{array}{lrcrcrcrcrcrcl} \text{Maximize} & 18x_1 & - & 7x_2 & + & 12x_3 & + & 5x_4 & & & + & 8x_6 & & \\ \text{Subject to:} & 2x_1 & - & 6x_2 & + & 2x_3 & + & 7x_4 & + & 3x_5 & + & 8x_6 & \leq & 1 \\ & -3x_1 & - & x_2 & + & 4x_3 & - & 3x_4 & + & x_5 & + & 2x_6 & \leq & -2 \\ & 8x_1 & - & 3x_2 & + & 5x_3 & - & 2x_4 & & & + & 2x_6 & \leq & 4 \\ & 4x_1 & & & + & 8x_3 & + & 7x_4 & - & x_5 & + & 3x_6 & \leq & 1 \\ & 5x_1 & + & 2x_2 & - & 3x_3 & + & 6x_4 & - & 2x_5 & - & x_6 & \leq & 5 \\ & & & & & & & & & x_1,x_2,\cdots,x_6 & & & \geq & 0 \end{array}$$

In this case, (9.26) says:

$$\begin{array}{rcrcrcrcrcr} 2y_1^* & - & 3y_2^* & + & 8y_3^* & + & 4y_4^* & + & 5y_5^* & = & 18 \\ -6y_1^* & - & y_2^* & - & 3y_3^* & & & + & 2y_5^* & = & -7 \\ 3y_1^* & + & y_2^* & & & - & y_4^* & - & 2y_5^* & = & 0 \\ & & y_2^* & & & & & & & = & 0 \\ & & & & & & & & y_5^* & = & 0 \end{array}$$

As the solution $(\frac{1}{3},0,\frac{5}{3},1,0)$ is a feasible solution of the dual problem (Problem 9.6), the proposed solution is optimal.

Secondly, is

$$x_1^* = 0,\ x_2^* = 2,\ x_3^* = 0,\ x_4^* = 7,\ x_5^* = 0$$

an optimal solution of the following problem?

$$\begin{array}{lrcrcrcrcrcl}
\text{Maximize} & 8x_1 & - & 9x_2 & + & 12x_3 & + & 4x_4 & + & 11x_5 & & \\
\text{Subject to:} & 2x_1 & - & 3x_2 & + & 4x_3 & + & x_4 & + & 3x_5 & \leq & 1 \\
& x_1 & + & 7x_2 & + & 3x_3 & - & 2x_4 & + & x_5 & \leq & 1 \\
& 5x_1 & + & 4x_2 & - & 6x_3 & + & 2x_4 & + & 3x_5 & \leq & 22 \\
& & & & & & & x_1, x_2, \cdots, x_5 & & & \geq & 0
\end{array}$$

Here (9.26) translates into:

$$\begin{array}{rcrcrcl}
-3y_1^* & + & 7y_2^* & + & 4y_3^* & = & -9 \\
y_1^* & - & 2y_2^* & + & 2y_3^* & = & 4 \\
 & & y_2^* & & & = & 0
\end{array}$$

As the unique solution of the system $(3.4, 0, 0.3)$ is not a feasible solution of Problem 9.6, the proposed solution is not optimal.

9.3.5 Interpretation of dual variables

As said in the introduction of this section, one of the major interests of the dual programme is that, in some problems, the variables of the dual problem have an interpretation.

A classical example is the *economical interpretation* of the dual variables of the following problem. Consider the problem that consits in maximizing the benefit of a company building some products. Each variable x_j of the primal problem measures the amount of product j that is built, and b_i the amount of resource i (needed to build the products) that is available. Note that, for any $i \leq n, j \leq m$, $a_{i,j}$ represents the number of units of resource i needed per unit of product j. Finally, c_j denotes the benefit (the price) of a unit of product j.

Hence, by checking the units of measure in the constraints $\sum a_{ij} y_i \geq c_j$, the variable y_i must represent a benefit per unit of resource i. Somehow, the variable y_i measures the unitary value of the resource i. This is illustrated by the following theorem the proof of which is omitted.

Theorem 9.13. *If Problem 9.5 admits a non degenerated optimal solution with value z^*, then there is $\varepsilon > 0$ such that, for any $|t_i| \leq \varepsilon$ $(i = 1, \ldots, m)$, the problem*

$$\begin{array}{lll}
\textit{Maximize} & \sum_{j=}^{n} c_j x_j & \\
\textit{Subject to} & \sum_{j=1}^{n} a_{ij} x_j \leq b_i + t_i & (i = 1, \ldots, m) \\
& x_j \geq 0 & (j = 1, \ldots, n)
\end{array}$$

admits an optimal solution with value $z^* + \sum_{i=1}^{m} y_i^* t_i$, where $(y_1^*, \ldots, y_m^*)$ is the optimal solution of the dual of Problem 9.5.

Theorem 9.13 shows how small variations in the amount of available resources can affect the benefit of the company. For any unit of extra resource i, the benefit increases by y_i^*. Sometimes, y_i^* is called the *marginal cost* of the resource i.

In many networks design problems, a clever interpretation of dual variables may help to achieve more efficient linear programme or to understand the problem better.

9.4 Exercices

9.4.1 General modelling

Exercise 9.1. Which problem(s) among P1, P2 and P3 are under the standard form?

$$\begin{array}{lrcrcr} \text{P1 : Maximize} & 3x_1 & - & 5x_2 & & \\ \text{Subject to:} & 4x_1 & + & 5x_2 & \geq & 3 \\ & 6x_1 & - & 6x_2 & = & 7 \\ & x_1 & + & 8x_2 & \leq & 20 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

$$\begin{array}{lrcrcrcrcrcr} \text{P2 : Minimize} & 3x_1 & + & x_2 & + & 4x_3 & + & x_4 & + & 5x_5 & & \\ \text{Subject to:} & 9x_1 & + & 2x_2 & + & 6x_3 & + & 5x_4 & + & 3x_5 & \leq & 5 \\ & 8x_1 & + & 9x_2 & + & 7x_3 & + & 9x_4 & + & 3x_5 & \leq & 2 \\ & & & & & & & x_1, x_2, x_3, x_4 & & & \geq & 0 \end{array}$$

$$\begin{array}{lrcrcr} \text{P3 : Maximize} & 8x_1 & - & 4x_2 & & \\ \text{Subject to:} & 3x_1 & + & x_2 & \leq & 7 \\ & 9x_1 & + & 5x_2 & \leq & -2 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

Exercise 9.2. Put under the standard form:

$$\begin{array}{lrcrcrcrcrcr} \text{P4 : Minimize} & -8x_1 & + & 9x_2 & + & 2x_3 & - & 6x_4 & - & 5x_5 & & \\ \text{Subject to:} & 6x_1 & + & 6x_2 & - & 10x_3 & + & 2x_4 & - & 8x_5 & \geq & 3 \\ & & & & & & & x_1, x_2, x_3, x_4, x_5 & & & \geq & 0 \end{array}$$

Exercise 9.3. Consider the following two problems corresponding to Problems 9.2 and 9.3 of the course. Prove that the first one is unfeasible and that the second one is unbounded.

$$\begin{array}{lrcrcr} \text{Maximize} & 3x_1 & - & x_2 & & \\ \text{Subject to:} & x_1 & + & x_2 & \leq & 2 \\ & -2x_1 & - & 2x_2 & \leq & -10 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

$$\begin{array}{lrcrcr} \text{Maximize} & x_1 & - & x_2 & & \\ \text{Subject to:} & -2x_1 & + & x_2 & \leq & -1 \\ & -x_1 & - & 2x_2 & \leq & -2 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

Exercise 9.4. Find necessary and sufficient conditions on the numbers s and t for the problem

$$\begin{array}{lrcrcr} \text{P5 : Maximize} & x_1 & + & x_2 & & \\ \text{Subject to:} & sx_1 & + & tx_2 & \leq & 1 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

a) to admit an optimal solution;

b) to be unfeasible;

c) to be unbounded.

Exercise 9.5. Prove or disprove: if the problem (9.1) is unbounded, then there exists an index k such that the problem:

$$\begin{array}{lrcll} \text{Maximize} & x_k & & & \\ \text{Subject to:} & \sum_{j=1}^{n} a_{ij}x_j & \leq & b_i & \text{for } 1 \leq i \leq m \\ & x_j & \geq & 0 & \text{for } 1 \leq j \leq n \end{array}$$

is unbounded.

Exercise 9.6. The factory RadioIn builds to types of radios A and B. Every radio is produced by the work of three specialists Pierre, Paul and Jacques. Pierre works at most 24 hours per week. Paul works at most 45 hours per week. Jacques works at most 30 hours per week. The resources necessary to build each type of radio and their selling prices as well are given in the following table:

	Radio A	Radio B
Pierre	1h	2h
Paul	2h	1h
Jacques	1h	3h
Selling prices	15 euros	10 euros

We assume that the company has no problem to sell its production, whichever it is.

a) Model the problem of finding a weekly production plan maximizing the revenue of RadioIn as a linear programme. Write precisely what are the decision variables, the objective function and the constraints.

b) Solve the linear programme using the geometric method and give the optimal production plan.

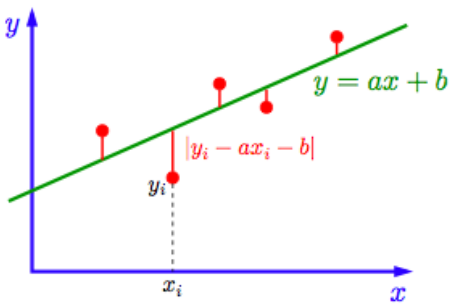
Exercise 9.7. The following table shows the different possible schedule times for the drivers of a bus company. The company wants that at least one driver is present at every hour of the working day (from 9 to 17). The problem is to determine the schedule satisfying this condition with minimum cost.

Time	9 – 11h	9 – 13h	11 – 16h	12 – 15h	13 – 16h	14– 17h	16 – 17h
Cost	18	30	38	14	22	16	9

Formulate an integer linear programme that solves the company decision problem.

Exercise 9.8 (Chebyshev approximation)**.** Data : m measures of points $(x_i, y_i) \in \mathbb{R}^{n+1}$, $i = 1, \ldots, m$.

Objective: Determine a linear approximation $y = ax + b$ minimizing the largest error of approximation. The decision variables of this problem are $a \in \mathbb{R}^n$ and $b \in R$. The problem may be



formulated as:

$$\min z = \max_{i=1,\dots,m} \{|y_i - ax_i - b|\}.$$

It is unfortunately not under the form of a linear program. Let us try to do some transformations.

Questions:

1. We call *Min-Max problem* the problem of minimizing the maximum of a set of numbers:

   $$\min z = \max\{c_1 x, \dots, c_k x\}.$$

   How to write a Min-Max problem as an LP?

2. Can we express the following constraints

   $$|x| \leq b$$

   or

   $$|x| \geq b$$

   in a LP (that is without absolute values)? If yes, how?

3. Rewrite the problem of finding a Chebyshev linear approximation as an LP.

9.4.2 Simplex

Exercise 9.9. Solve with the Simplex Method the following problems:

a.

$$\begin{array}{lrcrcrcl} \text{Maximize} & 3x_1 & + & 3x_2 & + & 4x_3 & & \\ \text{Subject to:} & & & & & & & \\ & x_1 & + & x_2 & + & 2x_3 & \leq & 4 \\ & 2x_1 & & & + & 3x_3 & \leq & 5 \\ & 2x_1 & + & x_2 & + & 3x_3 & \leq & 7 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

b.

$$\begin{array}{lrcrcrcrcl}
\text{Maximize} & 5x_1 & + & 6x_2 & + & 9x_3 & + & 8x_4 & & \\
\text{Subject to:} & & & & & & & & & \\
& x_1 & + & 2x_2 & + & 3x_3 & + & x_4 & \leq & 5 \\
& x_1 & + & x_2 & + & 2x_3 & + & 3x_4 & \leq & 3 \\
& & & & & & & x_1,x_2,x_3,x_4 & \geq & 0
\end{array}$$

c.

$$\begin{array}{lrcrcrcl}
\text{Maximize} & 2x_1 & + & x_2 & & & & \\
\text{Subject to:} & & & & & & & \\
& 2x_1 & + & 3x_2 & \leq & 3 \\
& x_1 & + & 5x_2 & \leq & 1 \\
& 2x_1 & + & x_2 & \leq & 4 \\
& 4x_1 & + & x_2 & \leq & 5 \\
& & & x_1,x_2 & \geq & 0
\end{array}$$

Exercise 9.10. Use the Simplex Method to describe *all* the optimal solutions of the following linear programme:

$$\begin{array}{lrcrcrcrcl}
\text{Maximize} & 2x_1 & + & 3x_2 & + & 5x_3 & + & 4x_4 & & \\
\text{Subject to:} & & & & & & & & & \\
& x_1 & + & 2x_2 & + & 3x_3 & + & x_4 & \leq & 5 \\
& x_1 & + & x_2 & + & 2x_3 & + & 3x_4 & \leq & 3 \\
& & & & & & & x_1,x_2,x_3,x_4 & \geq & 0
\end{array}$$

Exercise 9.11. Solve the following problems using the Simplex Method in two phases.

a.

$$\begin{array}{lrcrcrcl}
\text{Maximise} & 3x_1 & + & x_2 & & & \\
\text{Subject to:} & & & & & & \\
& x_1 & - & x_2 & \leq & -1 \\
& -x_1 & - & x_2 & \leq & -3 \\
& 2x_1 & + & x_2 & \leq & 4 \\
& & & x_1,x_2 & \geq & 0
\end{array}$$

b.

$$\begin{array}{lrcrcrcl}
\text{Maximise} & 3x_1 & + & x_2 & & & \\
\text{Subject to:} & & & & & & \\
& x_1 & - & x_2 & \leq & -1 \\
& -x_1 & - & x_2 & \leq & -3 \\
& 2x_1 & + & x_2 & \leq & 2 \\
& & & x_1,x_2 & \geq & 0
\end{array}$$

c.

$$\begin{array}{lrcrcl} \text{Maximise} & 3x_1 & + & x_2 & & \\ \text{Subject to:} & & & & & \\ & x_1 & - & x_2 & \leq & -1 \\ & -x_1 & - & x_2 & \leq & -3 \\ & 2x_1 & - & x_2 & \leq & 2 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

9.4.3 Duality

Exercise 9.12. Write the dual of the following linear programme.

$$\begin{array}{lrcrcl} \text{Maximize} & 7x_1 & + & x_2 & & \\ \text{Subject to:} & & & & & \\ & 4x_1 & + & 3x_2 & \leq & 3 \\ & x_1 & - & 2x_2 & \leq & 4 \\ & -5x_1 & - & 2x_2 & \leq & 3 \\ & & & x_1, x_2 & \geq & 0 \end{array}$$

Exercise 9.13. Consider the following linear programme.

$$\begin{array}{lrcrcrcrcl} \text{Minimize} & -2x_1 & - & 3x_2 & - & 2x_3 & - & 3x_4 & & \\ \text{Subject to:} & & & & & & & & & \\ & -2x_1 & - & x_2 & - & 3x_3 & - & 2x_4 & \geq & -8 \\ & 3x_1 & + & 2x_2 & + & 2x_3 & + & x_4 & \leq & 7 \\ & & & & & & & x_1, x_2, x_3, x_4 & \geq & 0 \end{array} \tag{9.27}$$

a) Write the programme (9.27) under the standard form.

b) Write the dual (D) of programme (9.27).

c) Give a graphical solution of the dual programme (D).

d) Carry on the first iteration of the Simplex Method on the linear programme (9.27).

After three iterations, one find that the optimal solution of this programme is $x_1 = 0$, $x_2 = 2$, $x_3 = 0$ and $x_4 = 3$.

e) Verify that the solution of (D) obtained at Question c) is optimal.

Exercise 9.14. We consider the following linear programme.

$$\begin{array}{lrcrcrcl} \text{Maximize} & x_1 & - & 3x_2 & + & 3x_3 & & \\ \text{Subject to :} & & & & & & & \\ & 2x_1 & - & x_2 & + & x_3 & \leq & 4 \\ & -4x_1 & + & 3x_2 & & & \leq & 2 \\ & 3x_1 & - & 2x_2 & - & x_3 & \leq & 5 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

If the solution $x_1^* = 0$, $x_2^* = 0$, $x_3^* = 4$ optimal?

Exercise 9.15. Prove that the following linear programme is unbounded.

$$\begin{array}{lrcrcrcr}
\text{Maximize} & 3x_1 & - & 4x_2 & + & 3x_3 & & \\
\text{Subject to :} & & & & & & & \\
& -x_1 & + & x_2 & + & x_3 & \leq & -3 \\
& -2x_1 & - & 3x_2 & + & 4x_3 & \leq & -5 \\
& -3x_1 & + & 2x_2 & - & x_3 & \leq & -3 \\
& & & & & x_1,x_2,x_3 & \geq & 0
\end{array}$$

Exercise 9.16. We consider the following linear programme.

$$\begin{array}{lrcrcrcrcrcr}
\text{Maximize} & 7x_1 & + & 6x_2 & + & 5x_3 & - & 2x_4 & + & 3x_5 & & \\
\text{Subject to:} & & & & & & & & & & & \\
& x_1 & + & 3x_2 & + & 5x_3 & - & 2x_4 & + & 2x_5 & \leq & 4 \\
& 4x_1 & + & 2x_2 & - & 2x_3 & + & x_4 & + & x_5 & \leq & 3 \\
& 2x_1 & + & 4x_2 & + & 4x_3 & - & 2x_4 & + & 5x_5 & \leq & 5 \\
& 3x_1 & + & x_2 & + & 2x_3 & - & x_4 & - & 2x_5 & \leq & 1 \\
& & & & & & & x_1,x_2,x_3,x_4,x_5 & & & \geq & 0.
\end{array}$$

Is the solution $x_1^* = 0, x_2^* = \frac{4}{3}, x_3^* = \frac{2}{3}, x_4^* = \frac{5}{3}, x_5^* = 0$, optimal?

Exercise 9.17. 1. Because of the arrival of new models, a salesman wants to sell off quickly its stock composed of eight phones, four hands-free kits and nineteen prepaid cards. Thanks to a market study, he knows that he can propose an offer with a phone and two prepaid cards and that this offer will bring in a profit of seven euros. Similarly, we can prepare a box with a phone, a hands-free kit and three prepaid cards, yielding a profit of nine euros. He is assured to be able to sell any quantity of these two offers within the availability of its stock. What quantity of each offer should the salesman prepare to maximize its net profit?

2. A sales representative of a supermarket chain proposes to buy its stock (the products, not the offers). What unit prices should he negociate for each product (phone, hands-free kits, and prepaid cards)?

Exercise 9.18 (FARKAS' LEMMA)**.** The following two linear programmes are duals of each other.

$$\begin{array}{lllll}
\text{maximize} & \mathbf{0x} & \text{subject to} & \mathbf{Ax} = \mathbf{0} & \text{and } \mathbf{x} \geq \mathbf{b} \\
\text{minimize} & -\mathbf{zb} & \text{subject to} & \mathbf{yA} - \mathbf{z} = \mathbf{0} & \text{and } \mathbf{z} \geq \mathbf{0}
\end{array}$$

Farkas' Lemma says that exactly one of the two linear systems:

$$\mathbf{Ax} = \mathbf{0},\ \mathbf{x} \geq \mathbf{b} \quad \text{and} \quad \mathbf{yA} \geq \mathbf{0},\ \mathbf{yAb} > 0$$

has a solution. Deduce Farkas' Lemma from the Duality Theorem (9.10).

Exercise 9.19. The following two linear programmes are duals of each other.

$$\begin{array}{lll} \text{minimize } \mathbf{y0} & \text{subject to} & \mathbf{yA} \geq \mathbf{c} \\ \text{maximize } \mathbf{cx} & \text{subject to} & \mathbf{Ax} = \mathbf{0} \quad \text{and } \mathbf{x} \geq \mathbf{0} \end{array}$$

A variant of Farkas' Lemma says that exactly one of the two linear systems:

$$\mathbf{yA} \geq \mathbf{c} \quad \text{and} \quad \mathbf{Ax} = \mathbf{0},\ \mathbf{x} \geq \mathbf{0},\ \mathbf{cx} > 0$$

has a solution. Deduce this variant of Farkas' Lemma from the Duality Theorem (9.10).

Exercise 9.20 (Application of duality to game theory- Minimax principle (*))**.** In this problem, based on a lecture of Shuchi Chawla, we present an application of linear programming duality in the theory of games. In particular, we will prove the Minimax Theorem using duality.

Let us first give some definition. A *two-players zero-sum game* is a protocol defined as follows: two players choose strategies in turn; given two strategies x and y, we have a *valuation function* $f(x,y)$ which tells us what the payoff for Player one is. Since it is a zero sum game, the payoff for the Player two is exactly $-f(x,y)$. We can view such a game as a matrix of payoffs for one of the players. As an example take the game of Rock-Paper-Scissors, where the payoff is one for the winning party or 0 if there is a tie. The matrix of winnings for player one will then be the following:

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

Where A_{ij} corresponds to the payoff for player one if player one picks the i-th element and player two the j-th element of the sequence (Rock, Paper, Scissors). We will henceforth refer to player number two as the column player and player number one as the row player. If the row player goes first, he obviously wants to minimize the possible gain of the column player.

What is the payoff of the row player? If the row player plays first, he knows that the column player will choose the minimum of the line he will choose. So he has to choose the line with the maximal minimum value. That is its payoff is

$$\max_i \min_j A_{ij}.$$

Similarly, what is the payoff of the column player if he plays first? If the column player plays first, the column player knows that the row player will choose the maximum of the column that will be chosen. So the column player has to choose the column with minimal maximum value. Hence, the payoff of the row player in this case is

$$\min_j \max_i A_{ij}.$$

Compare the payoffs. It is clear that

$$\max_i \min_j A_{ij} \leq \min_j \max_i A_{ij}.$$

The minimax theorem states that if we allow the players to choose probability distributions instead of a given column or row, then the payoff is the same no matter which player starts. More formally:

Theorem 9.14 (Minimax theorem)**.** *If x and y are probability vectors, then*

$$\max_y(\min_z y^T Ax) = \min_x(\max_y(y^T Ax)).$$

Let us prove the theorem.

1. Formulate the problem of maximizing its payoff as a linear programme.
2. Formulate the second problem of minimzing its loss as a linear programme.
3. Prove that the second problem is a dual of the first problem.
4. Conclude.

Exercise 9.21. Prove the following proposition.

Proposition 9.15. *The dual problem of the problem*

$$\textit{Maximize } \mathbf{c}^T\mathbf{x} \textit{ subject to } \mathbf{Ax} \leq \mathbf{a} \textit{ and } \mathbf{Bx} = \mathbf{b} \textit{ and } \mathbf{x} \geq \mathbf{0}$$

is the problem

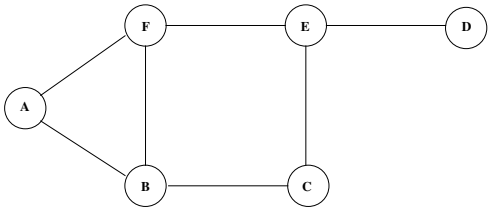
$$\textit{Minimize } \mathbf{a}^T\mathbf{y} + \mathbf{b}^T\mathbf{z} \textit{ subject to } \mathbf{A}^T\mathbf{y} + \mathbf{B}^T\mathbf{z} \geq \mathbf{c} \textit{ and } \mathbf{y} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0}.$$

9.4.4 Modelling Combinatorial Problems via (integer) linear programming

Lots of combinatorial problems may be formulated as linear programmes.

Exercise 9.22 (VERTEX COVER)**.** A *vertex cover* in a graph $G = (V,E)$ is a set K of vertices such that each edge e of E is incident to at least one vertex of K. The VERTEX COVER problem is to find a vertex cover of minimum cardinality in a given graph.

1. Express VERTEX COVER for the *following graph* as an integer linear programme:

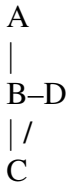


2. Express VERTEX COVER for a *general graph* as a linear programme.

Exercise 9.23 (EDGE COVER)**.** An *edge cover* of a graph $G = (V,E)$ is a set of edges $F \subseteq E$ such that every vertex $v \in V$ is incident to at least one edge of F. The EDGE COVER problem is to find an edge cover of minimum cardinality in a given graph.

Adapt the integer linear programme modelling VERTEX COVER to obtain an integer linear programming formulation of EDGE COVER.

Exercise 9.24. Consider the graph



What does the following linear programme do?

$$\begin{aligned} &\text{Minimize} && x_A + x_B + x_C + x_D \\ &\text{Subject to:} && \\ & && x_A + x_B \geq 1 \\ & && x_B + x_D \geq 1 \\ & && x_B + x_C \geq 1 \\ & && x_C + x_D \geq 1 \\ & && x_A \geq 0, x_B \geq 0, x_C \geq 0, x_D \geq 0 \end{aligned}$$

Exercise 9.25 (Maximum cardinality matching problem (Polynomial $<$ flows or augmenting paths))**.** Let $G = (V, E)$ be a graph. Recall that a *matching* $M \subseteq E$ is a set of edges such that every vertex of V is incident to at most one edge of M. The MAXIMUM MATCHING problem is to find a matching M of maximum size. Express MAXIMUM MATCHING as a integer linear programme.

Exercise 9.26 (Maximum clique (NP-complete))**.** Recall that a *clique* of a graph $G = (V, E)$ is a subset C of V, such that every two vertices in V are joined by an edge of E. The MAXIMUM CLIQUE problem consist of finding the largest cardinality of a clique.

Express MAXIMUM CLIQUE as an integer linear programme.

Exercise 9.27 (Resource assignment)**.** A university class has to go from Marseille to Paris using buses. There are some strong inimities inside the group and two people that dislike each other cannot share the same bus. What is the minimum number of buses needed to transport the whole group? Write a LP that solve the problem. (We suppose that a bus does not have a limitation on the number of places.)

Exercise 9.28 (French newspaper enigma)**.** What is the maximum size of a set of integers between 1 and 100 such that for any pair (a,b), the difference a-b is not a square ?

1. Model this problem as a graph problem.

2. Write a linear programme to solve it.

Exercise 9.29 (Maximum independent set (NP-hard))**.** An independent set of a graph G = (V, E) is a subset I of V , such that every two nodes in V are not joined by an edge of E. The maximum independent set problem consist of finding the largest cardinality of an independent set.

Exercise 9.30 (Minimum Set Cover (NP-hard))**.** *Input:* A universal set $U = \{1,...,n\}$ and a family $\mathcal{S}$ of subsets $S_1, \ldots, S_m$ of $\mathcal{U}$.
Optimization Problem: What is the smallest subset of subsets $\mathcal{T} \subset \mathcal{S}$ such that $\cup_{t_i \in \mathcal{T}} t_i = \mathcal{U}$?
Decision problem: Given an integer k, does there exists a subset of $\mathcal{T}$ of cardinality k, such that $\cup_{t_i \in \mathcal{T}} t_i = \mathcal{U}$? This decision problem is NP-complete.
Question: Write the set cover problem as a linear programme.
Alternative question: (easier, give the linear programme) Explain what is doing each line of the programme.

Exercise 9.31 (Example for the Maximum Set Packing)**.** Suppose you are at a convention of foreign ambassadors, each of which speaks English and other various languages.

- French ambassador: French, Russian
- US ambassador:
- Brazilian ambassador: Portuguese, Spanish
- Chinese ambassador: Chinese, Russian
- Senegalese ambassador: Wolof, French, Spanish

You want to make an announcement to a group of them, but because you do not trust them, you do not want them to be able to speak among themselves without you being able to understand them (you only speak English). To ensure this, you will choose a group such that no two ambassadors speak the same language, other than English. On the other hand you also want to give your announcement to as many ambassadors as possible.

Write a linear programme giving the maximum number of ambassadors at which you will be able to give the message.

Exercise 9.32 (Maximum Set Packing (Dual of the set cover problem))**.** Given a finite set S and a list of subsets of S.
Decision problem: Given an integer k, do there exist k pairwise disjoint sets (meaning, no two of them intersect)?
Optimization problem: What is the maximum number of pairwise disjoint sets in the list?

9.4.5 Modelling Flow Networks and Shortest Paths.

Definition 9.16 (Elementary flow network)**.** A flow network is a four-tuple $\mathcal{N} = (D,s,t,c)$ where

- $D = (V,A)$ is a directed graph with vertice set V and arc set A.
- c is a capacity function from A to $\mathbb{R}^+ \cup \infty$. For an arc $a \in A$, $c(a)$ represents its capacity, that is the maximum amount of flow it can carry.

- s and t are two distinct vertices: s is the source of the flow and t the sink.

A flow is a function f from A to $\mathbb{R}^+$ which respects the flow conservation constraints and the capacity constraints.

Exercise 9.33 (Maximum flow (Polynomial $<$ Ford-Fulkerson))**.** Write the linear program solving the maximum flow problem for a flow network.

Exercise 9.34 (Multicommodity flow)**.** Consider a flow network $\mathcal{N} = (D, s, t, c)$. Consider a set of demands given by the matrix $\mathcal{D} = (d_{ij} \in \mathbb{R}; i, j \in V, i \neq j)$, where d_{ij} is the amount of flow that has to be sent from node i to node j. The multicommodity flow problem is to determine if all demands can be routed simultaneously on the network. This problem models a telecom network and is one of the fundamental problem of the networking research field.

Write a linear program that solves the multicommodity flow problem.

Exercise 9.35 ($s-t$ shortest path)**.** Let $D = (V, A, l)$ be a weighted digraph. l is a length function from A to $\mathbb{R}^+$. For $a \in A$, l(a) is the length of arc a. Let s and t two distinguised vertices.

Write a linear program that finds the length of a shortest path between s and t.

Exercise 9.36 (How far are you from anybody in Facebook?)**.** In graph theory, the *distance* between two vertices in a graph is the number of edges in a shortest path connecting them. We consider the graph of Facebook members. Two people are at distance one if they are friends.

The *eccentricity* ε of a vertex v is the greatest distance between v and any other vertex. It can be thought of as how far a node is from the node most distant from it in the graph. The *diameter* of a graph is the maximum eccentricity of any vertex in the graph. That is, it is the greatest distance between any pair of vertices.

1. Write an LP to compute the eccentricity of a given vertex.

2. Write an LP which computes the diameter of the Facebook graph.

Exercise 9.37 (Minimum Cut Problem)**.**

Definition 9.17 (Cut - Reminder)**.** In a flow network $\mathcal{N} = (G, s, p, c)$ a cut is a bipartition $C = (V_s, V_p)$ of the vertices of G such that $s \in V_s$ and $p \in V_p$. The capacity of the cut C, denoted by $\delta(C)$, is the sum of the capacities of the out-arcs of V_s (i.e., the arcs (u, v) with $u \in V_s$ and $v \in V_p$).

Write a linear program that solves the minimum cut problem.

Hint: Use variables to know in which partition is each vertex and additional variables to know which edges are in the cut.

Bibliography

[1] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1998.

[2] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[3] *ILOG CPLEX optimization software*. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`.

[4] *GNU Linear Programming Kit*. `http://www.gnu.org/software/glpk/`.

[5] D. Gale, H. W. Kuhn, and A. W. Tucker. Linear Programming and the Theory of Games. In *T. C. Koopmans (ed.), Activity Analysis of Production and Allocation*, New York, Wiley, 1951.

[6] M. Sakarovitch. *Optimisation Combinatoire: Graphes et Programmation Linéaire*. Hermann, Paris, 1984.

Chapter 10

Polynomiality of Linear Programming

In previous sections, we have seen the simplex method for solving linear programmes which appears to be very efficient in practice. While it is known that Gaussian elimination can be implemented in polynomial time, the number of pivot rules used throughout the simplex method may be exponential. More precisely, Klee and Minty gave an example of a linear programme such that the simplex method goes through each of the 2^n extreme points of the corresponding polytope [1].

In this section, we survey two methods for solving linear programmes in polynomial time. On the one hand, the *Ellipsoid Method* [2, 3] is not competitive with the simplex method in practice but it has important theoretical side-effects. On the other hand, the *Interior Point Methods* compete with the simplex method in practice.

First of all, we define the *input size* of a linear programme. Recall that an integer $i \in \mathbb{Z}$ can be encoded using $< i >= \lceil \log_2(|i|+1) \rceil + 1$ bits. For a rational number $r = p/q \in \mathbb{Q}$, the size of r is $< r >=< p > + < q >$. Similarly, any rational matrix can be encoded using $< A >= \sum_{i=1}^{m} \sum_{j=1}^{n} < a_{i,j} >$ bits. Also, multiplying two integers a and b runs in time $O(< a > + < b >)$.

In what follows, we consider the linear programme L defined by:

$$\begin{array}{rl} \text{Maximize} & \mathbf{c}^T\mathbf{x} \\ \text{Subject to:} & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \tag{10.1}$$

We restrict ourselves to the case when $\mathbf{A} \in \mathbb{Q}^{m \times n}$, $\mathbf{b} \in \mathbb{Q}^m$ and $\mathbf{c} \in \mathbb{Q}^n$ have rational coefficients. Therefore, the input size of L is $< L >=< A > + < b > + < c >$. Say differently, $< L >$ is a polynomial in n, m and $< B >$ where B is the largest coefficient in $\mathbf{A}, \mathbf{b}$ and $\mathbf{c}$.

The two methods presented in this section are polynomial in $< L >$. It is a long standing open problem to know whether linear programming is strongly polynomial, i.e., whether there exists an algorithm that solves a linear programme and running in time polynomial in n and m.

The interest of the ellipsoid method comes from the fact that, in particular cases, it works independently from the number m of constraints. More precisely, if we are given a *separation oracle* that, given a vector $\mathbf{x}$, answers that $\mathbf{x}$ satisfies $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ or returns an inequality not satisfied by $\mathbf{x}$, then the ellipsoid method works in time polynomial in n and $< B >$.

10.1 Ellipsoid Method

The ellipsoid method has been proposed in the 70s by Shor, Judin and Nemirovski for solving some nonlinear optimization problems. In 1979, Khachyian showed how to use it for solving linear programmes.

10.1.1 Optimization versus faisibility

We first recall (or state) some definitions and results of linear algebra.

A *convex set* $C \subseteq \mathbb{R}^n$ is such that $\forall \mathbf{x}, \mathbf{y} \in C$ and $\forall 0 \leq \lambda \leq 1$, $\lambda\mathbf{x} + (1-\lambda)\mathbf{y} \in C$. An *half space* H is a (convex) set $\{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{c}^T x \leq \delta\}$ with $\mathbf{c} \in \mathbb{R}^n$, $\delta \in \mathbb{R}$. A closed convex set is the intersection of a family of half spaces. A *polyhedron* $\mathcal{K}$ is a closed convex set that is the intersection of a finite family of half spaces, i.e., $\mathcal{K} = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$. A *polytope* is the convex hull of a finite set $X \subseteq \mathbb{R}^n$, i.e., $\{\mathbf{x} \in \mathbb{R}^n \ : \ \sum \lambda_i \mathbf{x}_i, \ \sum_i \lambda_i \leq 1, \mathbf{x}_i \in X\}$.

Theorem 10.1. *A set is polytope if and only if it is a bounded polyhedron.*

Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, the system of inequalities $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ is *feasible* if there is $\mathbf{x} \in \mathbb{R}^n$ that satisfies it, i.e., if the polyhedron $\{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ is not empty. We say that the system is *bounded* if there is $R \in \mathbb{R}$ such that the set of the solutions of the system is included in a ball of radius $\leq R$ in $\mathbb{R}^n$.

The ellispoid method aims at deciding whether a polytope is not empty and, if possible, at finding some vector in it. We first show that it is sufficient to solve linear programmes. In other words, the next theorem shows that solving a linear programme can be reduced to the feasibility of a system of linear inequalities.

Theorem 10.2. *If it can be decided in polynomial time whether a system of linear inequalities is feasible then linear programmes can be solved in polynomial time.*

Proof. Consider the linear programme L described in 10.1. By the strong duality theorem (Theorem 9.10), L admits an optimal solution if and only if the system $\{\mathbf{x} \geq 0, \ \mathbf{y} \geq 0, \ \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{A}^\mathbf{T}\mathbf{y} \geq \mathbf{c}, \mathbf{c}\mathbf{x} \geq \mathbf{b}^\mathbf{T}\mathbf{y}\}$ is feasible and bounded, i.e., it is a non empty polytope. □

Therefore, from now on, we focus on the feasibility of the bounded system

$$\begin{array}{c} \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ \mathbf{A} \in \mathbb{Q}^{m \times n}, \mathbf{b} \in \mathbb{Q}^m \end{array} \tag{10.2}$$

10.1.2 The method

In this section, we describe the Ellipsoid Method. Given a polytope $\mathcal{K}$ and $\mathcal{V} \in \mathbb{R}$, this method either returns a vector $\mathbf{x} \in \mathcal{K}$ or states that $\mathcal{K}$ has volume less than $\mathcal{V}$.

Definitions and notations

A matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ is *positive definite* if and only if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for any $\mathbf{x} \in \mathbb{R}^n \setminus \{0\}$, or equivalently, $\mathbf{A} = \mathbf{Q}^T diag(\lambda_1, \cdots, \lambda_n)\mathbf{Q}$ where $\mathbf{Q} \in \mathbb{R}^{n\times n}$ is an orthogonal matrix and $\lambda_i > 0$ for any $i \leq n$. Another equivalent definition is that $\mathbf{A}$ is positive definite if $\mathbf{A} = \mathbf{B}^T\mathbf{B}$ where $\mathbf{B}$ is triangular with strictly positive diagonal elements.

The *unit ball* $\mathcal{B}(0,1)$ is $\{\mathbf{x} \in \mathbb{R}^n \ : \ ||\mathbf{x}|| \leq 1\}$ with volume V_n.

An *ellipsoid*, denoted by $\varepsilon(\mathbf{A},\mathbf{b})$, is the image of $\mathcal{B}(0,1)$ under a linear map $t : \mathbb{R}^n \to \mathbb{R}^n$ such that $t(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is an invertible matrix and $\mathbf{b} \in \mathbb{R}^n$. That is, $\varepsilon(\mathbf{A},\mathbf{b}) = \{\mathbf{x} \in \mathbb{R}^n \ : \ ||\mathbf{A}^{-1}\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}|| \leq 1\}$. Alternatively, the ellipsoid $\varepsilon(\mathbf{A},\mathbf{b})$ can be defined as $\{\mathbf{x} \in \mathbb{R}^n \ : \ (\mathbf{x}-\mathbf{b})^T\mathbf{M}(\mathbf{x}-\mathbf{b}) \leq 1\}$ where $\mathbf{M} = (\mathbf{A}^{-1})^T\mathbf{A}^{-1}$ is positive definite.

Proposition 10.3. *The volume* $vol(\varepsilon(\mathbf{A},\mathbf{b}))$ *of* $\varepsilon(\mathbf{A},\mathbf{b})$ *is* $|det(A)|.V_n$.

The ellipsoid method.

Let $\mathcal{K} = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ be a polytope with $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and let $\mathcal{V} \in \mathbb{R}$. Assume we are given $\mathbf{M_0} \in \mathbb{R}^{n\times n}$ and $\mathbf{c_0} \in \mathbb{R}^n$ such that $\mathcal{K} \subseteq \varepsilon_0(\mathbf{M_0}, \mathbf{c_0})$. The ellipsoid method proceeds as follows.

Algorithm 10.1 (Ellipsoid Method)**.**

1. Let $k = 0$. Note that $\mathcal{K} \subseteq \varepsilon_k(\mathbf{M}_k, \mathbf{c}_k)$;
2. If $vol(\varepsilon_k(\mathbf{M}_k, \mathbf{c}_k)) < \mathcal{V}$ then stop;
3. Otherwise, if $\mathbf{c}_k \in \mathcal{K}$ then return $\mathbf{c}_k$;
4. Else let $\mathbf{a}_i^T\mathbf{x} \leq b_i$ be an inequality defining $\mathcal{K}$, i.e., $\mathbf{a}_i$ is a row of $\mathbf{A}$, such that $\mathbf{a}_i^T\mathbf{c}_k > b_i$;

   Let $\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1})$ be an ellipsoid with volume $\leq e^{-\frac{1}{2(n+1)}}.vol(\varepsilon_k(\mathbf{M}_k, \mathbf{c}_k))$ such that

   $$\varepsilon_k(\mathbf{M}_k, \mathbf{c}_k) \cap \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{a}_i^T\mathbf{x} \leq \mathbf{a}_i^T\mathbf{c}_k\} \subseteq \varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1});$$

5. $k \leftarrow k+1$ and go to step 2.

Theorem 10.4. *The ellipsoid method computes a point in* $\mathcal{K}$ *or asserts that* $vol(\mathcal{K}) < \mathcal{V}$ *in at most* $2.n.\ln\frac{vol(\varepsilon_0(\mathbf{M_0},\mathbf{c_0}))}{\mathcal{V}}$ *iterations.*

Proof. After k iterations, $vol(\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1}))/vol(\varepsilon_0(\mathbf{M_0}, \mathbf{c_0})) \leq e^{-\frac{k}{2(n+1)}}$. Since we stop as soon as $vol(\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1})) < \mathcal{V}$, there are at most $2.n.\ln\frac{vol(\varepsilon_0(\mathbf{M_0},\mathbf{c_0}))}{\mathcal{V}}$ iterations. □

Discussion about the complexity and separation oracle.

Let $\mathcal{K}$ be the rational polytope defined in equation 10.2. Let B be the largest absolute value of the coefficients of $\mathbf{A}$ and $\mathbf{b}$. We now show that the ellipsoid method can be implemented in time polynomial in $< \mathcal{K} >$, i.e., in n, m and B.

Here, we first discuss the main steps of the proof, the technical results needed for this purpose are postponed in next subsection.

- First, a lower bound $\mathcal{V}$ on the volume of $\mathcal{K}$ must be computed. Theorem 10.5 defines such a $\mathcal{V}$ in the case $\mathcal{K}$ is full dimensional.

  Otherwise, there is a first difficulty. Theorem 10.5 shows that there is $\varepsilon > 0$, such that $\mathcal{K}$ is empty if and only if $\mathcal{K}' = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b} + \mathbf{U}\}$ is empty, where $< \varepsilon >$ is polynomial in n, B and $\mathbf{U} \in \mathbb{R}^m$ is the vector with all coordinates equal to ε. Moreover, $\mathcal{K}'$ is full dimensional (see Exercise 10.3).

  Therefore, the ellipsoid method actually applies on the polytope $\mathcal{K}'$. If $\mathcal{K}'$ is empty, then $\mathcal{K}$ is empty as well. Otherwise, the ellipsoid method returns $\mathbf{x}' \in \mathcal{K}'$ and the solution $\mathbf{x}'$ is rounded to a solution $\mathbf{x} \in \mathcal{K}$. We do not detail this latter operation in this note.

- Then, an initial ellipsoid $\varepsilon_0(\mathbf{M_0}, \mathbf{c_0})$ containing $\mathcal{K}$ is required. Theorem 10.6 describes how to define it in such a way that $< \varepsilon_0(\mathbf{M_0}, \mathbf{c_0}) >$ is polynomial in n and B.

- The crucial step of the ellipsoid method is step 4. Theorem 10.8 proves that the desired ellipsoid $\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1})$ always exists. Moreover, it can be computed from $\varepsilon_k(\mathbf{M}_k, \mathbf{c}_k)$ and the vector $\mathbf{a}_i$, with a number of operations that is polynomial in $< \varepsilon_k(\mathbf{M}_k, \mathbf{c}_k) >$ and $< \mathbf{a}_i >$.

  Another technicality appears here. Indeed, following Theorem 10.8, some square-roots are needed when defining $\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1})$. Therefore, its encoding might be not polynomial in $< \varepsilon_k(\mathbf{M}_k, \mathbf{c}_k) >$ and $< \mathbf{a}_i >$. It is actually possible to ensure that $\varepsilon_{k+1}(\mathbf{M}_{k+1}, \mathbf{c}_{k+1})$ satisfies the desired properties and can be encoded polynomially in $< \varepsilon_k(\mathbf{M}_k, \mathbf{c}_k) >$ and $< \mathbf{a}_i >$. We do not give more details in this note.

  Since, by the previous item, $< \varepsilon_0(\mathbf{M_0}, \mathbf{c_0}) >$ is polynomial in n and B, therefore, for any $k > 0$, $< \varepsilon_k(\mathbf{M_k}, \mathbf{c_k}) >$ and the computation of the ellipsoid are polynomial in n and B.

- Now, using previous item and Proposition 10.3, step 2 is clearly polynomial in n and B.

- Finally, let us consider the following question. Given a system $\mathcal{K} = \{\mathbf{x} \in \mathbb{Q}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ with $\mathbf{A} \in \mathbb{Q}^{m \times n}$, $\mathbf{b} \in \mathbb{Q}^m$ and a vector $\mathbf{y} \in \mathbb{Q}^n$, decide if $y \in \mathcal{K}$ or returns an index $i \leq m$ such that $\mathbf{a}_i^T \mathbf{c}_k > b_i$. Clearly, this can be decided in time polynomial in m, n and B (the largest absolute value of the coefficients) by simply checking the inequalities one by one.

The above discussion proves that the ellipsoid method runs in time polynomial in m, n and B. Moreover, note that, m appears in the complexity only when solving the question of the last item (step 3). The main interest of the ellipsoid method is that if we are given an oracle for answering this question independently of m, then the ellipsoid method runs in time polynomial

in n and B (independently of m). Even more, it is not necessary to explicitly know all constraints (see Exercise 10.4).

10.1.3 Complexity of the ellipsoid method

In this section, we formally state (and prove some of) the Theorems used in the analysis of the ellipsoid method (see discussion above).

In the following, for any $\mathbf{x} \in \mathbb{R}^n$, $x(i)$ denotes the i^{th} coordinate of $\mathbf{x}$.

Initialization: finding the lower bound

The next theorem allows to find a lower bound for the volume of the considered polytope, if it is not empty.

Theorem 10.5. *Let $\mathcal{K} = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ be a polytope and let B be the largest absolute value of the coefficients of $\mathbf{A}$ and $\mathbf{b}$.*

- *if $\mathcal{K}$ is full dimensional, its volume is lower bounded by $1/(nB)^{3n^2}$;*
- *let $\varepsilon = 1/((n+1).(nB)^n)$ and let $\mathcal{K}' = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b} + \mathbf{U}\}$ where $\mathbf{U}$ is the vector with all components equal to ε. Then, $\mathcal{K}'$ is full dimensional and is not empty if and only if $\mathcal{K}$ is not empty*

Proof. TBD □

Initialization: finding the initial ellipsoid

The next theorem allows to define the initial ellipsoid $\varepsilon_0(\mathbf{A_0}, \mathbf{a_0})$.

Theorem 10.6. *Let $\mathcal{K} = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ be a polytope and let B be the largest absolute value of the coefficients of $\mathbf{A}$ and $\mathbf{b}$. Then $\mathcal{K}$ is contained into the ball $\mathcal{B}(0, n^n B^n)$.*

Proof. The *vertices* of a convex set are the points of it that are not a linear combination of any other vectors in this convex set. Equivalently, a vertex of $\mathcal{K}$ is determined as the unique solution $\mathbf{v}$ of a linear system $\mathbf{A}'\mathbf{x} = \mathbf{b}'$ where $\mathbf{A}'\mathbf{x} \leq \mathbf{b}'$ is a subsystem of $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ and $\mathbf{A}'$ is nonsingular.

By the Cramer's formula, the i^{th} coefficient of such a solution $\mathbf{v}$ is $v_i = det(\mathbf{A}'_i)/det(\mathbf{A}')$ where $\mathbf{A}'_i$ is the matrix obtained from $\mathbf{A}'$ by replacing its i^{th} column by the vector $\mathbf{b}$. The Hadamard inequality states that $|det(\mathbf{M})| \leq \Pi_{i=1}^n ||\mathbf{m}_i||$ for any $\mathbf{M} \in \mathbb{R}^{n \times n}$ with columns $\mathbf{m}_i$, $i \leq n$. Therefore, $|det(\mathbf{A})|$, $|det(\mathbf{A}')|$, $|det(\mathbf{A}'_i)|$ and $|v_i|$ are all upper bounded by $n^{n/2}B^n$.

Hence, any vertex of $\mathcal{K}$ lies in $\{\mathbf{x} \in \mathbb{R}^n \ : \ ||\mathbf{x}|| \leq n^{n/2}B^n\}$. To conclude it is sufficient to recall that a polytope equals the convex hull of its vertices. □

Computing the next ellipsoid containing $\mathcal{K}$ (step 4)

Lemma 10.7. *The half-unit ball* $\mathcal{B}_{1/2} = \{\mathbf{x} \in \mathbb{R}^n \ : \ ||\mathbf{x}|| \leq 1, x(1) \geq 0\}$ *is contained in the ellipsoid* $E = \varepsilon(\mathbf{A}, \mathbf{b})$ *with volume at most* $V_n.e^{-\frac{1}{2(n+1)}}$ *where* $\mathbf{A} = diag(\frac{n}{n+1}, \sqrt{\frac{n^2}{n^2-1}}, \cdots, \sqrt{\frac{n^2}{n^2-1}})$ *and* $\mathbf{b} = (1/(n+1), 0, \cdots, 0)$.

Proof. First, $E = \{\mathbf{x} \in \mathbb{R}^n \ : \ ||\mathbf{A}^{-1}\mathbf{x} - \mathbf{A}^{-1}\mathbf{b}||^2 = (\frac{n+1}{n})^2(x(1) - \frac{1}{n+1})^2 + \frac{n^2-1}{n^2}\sum_{i=2}^n x(i)^2 \leq 1\}$.

Let $\mathbf{y} \in \mathcal{B}_{1/2}$, we show that $\mathbf{y} \in E$ and then $\mathcal{B}_{1/2} \subseteq E$.

$$\begin{array}{rcl} & (\frac{n+1}{n})^2(y(1) - \frac{1}{n+1})^2 + \frac{n^2-1}{n^2}\sum_{i=2}^n y(i)^2 & \\ = & \frac{2n+2}{n^2}y(1)(y(1)-1) + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=2}^n y(i)^2 & \\ \leq & \frac{1}{n^2} + \frac{n^2-1}{n^2} & \leq 1 \end{array}$$

Moreover, the volume of E is $|det(A)|.V_n$ and $det(A) = \frac{n}{n+1}(\frac{n^2}{n^2-1})^{(n-1)/2}$. Using the fact that $1 + x \leq e^x$, we obtain that $det(A) \leq e^{-1/(n+1)}e^{(n-1)/(2(n^2-1))} = e^{-\frac{1}{2(n+1)}}$. □

Theorem 10.8. *The half-ellipsoid* $\varepsilon(\mathbf{A}, \mathbf{a}) \cap \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{c}^T\mathbf{x} \leq \mathbf{c}^T\mathbf{a}\}$ *is contained in the ellipsoid* $\varepsilon(\mathbf{A}', \mathbf{a}')$ *where*

$$\mathbf{a}' = \mathbf{a} - \frac{1}{n+1}\mathbf{b} \quad \textit{and} \quad \mathbf{A}' = \frac{n^2}{n^2-1}(\mathbf{A} - \frac{2}{n+1}\mathbf{b}\mathbf{b}^T) \quad \textit{where} \quad \mathbf{b} = \mathbf{A}\mathbf{c}/\sqrt{\mathbf{c}^\mathbf{T}\mathbf{A}\mathbf{c}}$$

Moreover, the ratio $vol(\varepsilon(\mathbf{A}', \mathbf{a}'))/vol(\varepsilon(\mathbf{A}, \mathbf{a})) \leq e^{-1/(2(n+1))}$.

Proof. This follows from the fact that the half-ellipsoid is the image of the half-unit ball under a linear transformation and from Lemma 10.7. Also, the ratio of the volumes of the two ellipsoids is invariant under the linear transformation. □

Note that the square root in the definition of $\varepsilon(\mathbf{A}', \mathbf{a}')$ implies that this ellipsoid may not be computed exactly. Nevertheless, it can be modified so that the intermediate results are rounded using a number of bits that is polynomial in $< \mathcal{K} >$. A suitable choice of the rounding constants ensures that the obtained rational ellipsoid still contains $\mathcal{K}$.

10.2 Interior Points method

The interior point method actually gather a family of methods. The approach consists in walking through the interior of the set of feasible solutions (while the simplex method walks along the boundary of this the polytope from vertex to vertex, and the ellipsoid method encircles this polytope).

Among the family of interior points method, Karamakar proposed an algorithm and proved it performs polynomially to solve linear programmes.

It is important to know that the interior points method typically outperforms the simplex method on very large problems.

10.3 Exercises

Exercise 10.1. Prove Theorem 10.1

Exercise 10.2. Prove Theorem 10.3

Exercise 10.3. Let $P = \{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{Ax} \leq \mathbf{b}\}$ be a polyhedron and $\varepsilon > 0$. Let $\mathbf{U} \in \mathbb{R}^m$ be the vector with all coordinates equal to ε. Shows that $\{\mathbf{x} \in \mathbb{R}^n \ : \ \mathbf{Ax} \leq \mathbf{b} + \mathbf{U}\}$ is full dimensional.

Exercise 10.4. Minimum cost arborescence. Let D be a n-node directed graph with each arc $a \in A(D)$ has weight c_a and let $r \in V(D)$. Consider the following problem:

$$\begin{array}{rll} \text{Minimize} & \sum_{a \in A(D)} c_a x_a & \\ \text{Subject to:} & \sum_{a \in \delta^-(S)} x_a \geq 1 & \forall S \subseteq V(D) \setminus \{r\} \\ & x_a \geq 0 & \forall a \in A(D) \end{array}$$

Give an algorithm that decide whether $\mathbf{x} \in \mathbb{R}^n$ is a feasible solution or returns a certificate $S \subseteq V(D) \setminus \{r\}$ that $\mathbf{x}$ is not feasible, in time polyomial in n. Conclusion?

Bibliography

[1] V. Klee and G.J. Minty. How Good is the Simplex Algorithm? In O. Shisha, editor, Inequalities, III, pages 159-175. Academic Press, New York, NY, 1972.

[2] L. G. Khachiyan. A polynomial algorithm in linear programming. Doklady Akademii Nauk SSSR 244, pages 1093-1096, 1979. (English translation: Soviet. Math Dokl. 20, 191-194)

[3] M. Grötschel, L. Lovász and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. Combinatorica 1(2): 169-197 (1981)

[4] V. Chvatal. Linear Programming. W.H. Freeman & Company, 1983

[5] J. Matousek and B. Gärdner. Understanding and Using Linear Programming. Springer, 2007

Chapter 11

Fractional Relaxation

Lots of combinatorial problems are very close to linear programmes. Indeed they can be formulated as a linear programme with the extra requirement that some of the variable be integers (and not real). Such programmes are called *integer linear programmes*.

For example, consider MAXIMUM MATCHING, which is the problem of finding the largest matching in a graph. Recall that a *matching* in a graph is a set of pairwise non-adjacent edges. The size of a largest matching in a graph G is denoted by $\mu(G)$. With any matching M of a graph G, we may associate its $(0,1)$-valued indicator vector $\mathbf{x}$ (that is $x_e = 1$ if $e \in M$ and $x_e = 0$ otherwise). Since no matching has more than one edge incident with any vertex, every such vector $\mathbf{x}$ satisfies the constraint $\sum_{e \ni v} x_e \leq 1$, for all $v \in V$. Thus an instance of MAXIMUM MATCHING may be formulated as the following integer linear programme.

$$\begin{array}{l} \text{Maximize } \sum_{e \in E} x_e \text{ subject to :} \\ \\ \begin{array}{rl} \sum_{e \ni v} x_e \leq 1 & \text{for all } v \in V \\ x_e \in \mathbb{N} & \text{for all } e \in E \end{array} \end{array} \tag{11.1}$$

For every edge e, the constraint in any of its endvertices implies that $x_e \leq 1$. So the condition $x_e \in \mathbb{N}$ is equivalent to the condition $x_e \in \{0,1\}$.

As a second example, consider VERTEX COVER, the problem of finding a minimum vertex cover in a graph. Recall that a *vertex cover* in a graph $G = (V,E)$ is a set of vertices $S \subset V$ such that every edge has at least one endvertex in S. A vertex cover of minimum cardinality is said to be *minimum*. The cardinality of a minimum vertex cover is denoted by $\nu(G)$. Letting $\mathbf{y}$ be the indicator vector of a vertex cover (i.e. $y_v = 1$ if it is in the vertex cover and $y_v = 0$ otherwise), an instance of VERTEX COVER has following Integer Linear Programming formulation.

$$\begin{array}{l} \text{Minimize } \sum_{v \in V} y_v \text{ subject to :} \\ \\ \begin{array}{rl} y_u + y_v \geq 1 & \text{for all } uv \in E \\ y_v \in \mathbb{N} & \text{for all } v \in V \end{array} \end{array} \tag{11.2}$$

Again the condition $\mathbf{y}$ integral is equivalent to the condition $\mathbf{y}$ $(0,1)$-valued. Indeed if $y_v > 1$ for some v, then setting y_v to 1, we obtain a solution with smaller weight, so an optimal solution is

necessarily $(0,1)$-valued.

VERTEX COVER is known to be NP- hard (see [10]) and, like many NP- hard problems (see e.g. Exercise 11.1 and 11.2), it may be formulated as integer linear programme. Therefore, solving integer linear programmes is NP- hard in general. However, writing a problem as an integer linear programme is often very useful. First, one can always relax the integrality conditions to obtain a linear programme, called *fractional relaxation* of problem.

For instance, the fractional relaxation of (11.1) is the following linear programme.

$$\begin{array}{l} \text{Maximize } \sum_{e\in E} x_e \text{ subject to :} \\ \\ \begin{array}{rl} \sum_{e\ni v} x_e \leq 1 & \text{for all } v \in V \\ x_e \geq 0 & \text{for all } e \in E \end{array} \end{array} \tag{11.3}$$

Since a feasible solution of an integer linear programme is always a feasible solution of its fractional relaxation, the optimal solution of this linear programme is at least as good as that of the integer linear programme. That is, in a maximization problem, the relaxed programme has a value greater than or equal to that of the original programme, while in a minimization problem the relaxed programme has a value smaller than or equal to that of the original programme. For instance, a feasible solution to (11.3), is called a *fractional matching*, and the maximum value of a fractional matching of G is called the *fractional matching number* and is denoted $\mu_f(G)$. Then $\mu_f(G) \geq \mu(G)$. For many graphs G, we have $\mu_f(G) > \mu(G)$. For example, for any odd cycle C_{2k+1}. Setting $x_e = 1/2$ for all edge e, we obtain that $\mu_f(G) \geq (2k+1)/2$ (in fact, $\mu_f(G) = (2k+1)/2$, see Exercise 11.3) while $\mu(G) = k$.

The problem FRACTIONAL MATCHING, which consists in finding a fractional maximal of value $\mu_f(G)$ can be formulated as the linear programme (fractional-matching) which has a polynomial number of constraints. Therefore, it can be solved in polynomial time.

Similarly, the fractional relaxation of (11.2) is

$$\begin{array}{l} \text{Minimize } \sum_{v\in V} y_v \text{ subject to :} \\ \\ \begin{array}{rl} y_u + y_v \geq 1 & \text{for all } uv \in E \\ y_v \geq 0 & \text{for all } v \in V \end{array} \end{array} \tag{11.4}$$

A feasible solution to (11.4), is called a *fractional vertex cover*, and the minimum value of a fractional cover of G is called the *fractional vertex cover number* and is denoted $\nu_f(G)$. Then $\nu_f(G) \leq \nu(G)$. For many graphs G, we have $\nu_f(G) < \nu(G)$. For example, for any odd cycle C_{2k+1}. Setting $y_v = 1/2$ for all vertex v, we obtain that $\nu_f(G) \leq (2k+1)/2$ while $\nu(G) = k+1$. The problem FRACTIONAL VERTEX COVER, which consists in finding a fractional vertex cover of value $\nu_f(G)$ can be solved in polynomial time, because of its linear programming formulation (fractional-cover).

Observe that the two linear programmes (11.3) and (11.4) are dual to each other. Thus by the Duality Theorem, $\mu_f(G) = \nu_f(G)$, and so

$$\mu(G) \leq \mu_f(G) = \nu_f(G) \leq \nu(G).$$

Hence, fractional relaxation and the Duality Theorem prove the (simple) fact that in a graph the maximum size of a matching is smaller than the minimum size of a vertex cover. More generally, it may be used to prove some relation between two parameters which may seem unrelated at first glance, and may lead to nice and elegant proofs. See Subsection 11.1.3.

But fractional relaxation is also very useful in an algorithmic prospect. Indeed sometimes the fractional problem has the same optimal value as the original one, and so solving the former via linear programming yields a polynomial-time algorithm to solve the later. This is the case for MAXIMUM MATCHING in bipartite graphs.

Proposition 11.1. *If G is bipartite, then (11.3) has an integral optimal solution, which is thus an optimal solution to (11.1). So $\mu(G) = \mu_f(G)$.*

Proof. Let $\mathbf{x}$ be an optimal solution to (11.3). Then $\mu_f(G) = \sum_{e \in E} x_e$.

If $\mathbf{x}$ is integral, then we are done. If not, we describe a procedure that yields another optimal solution with strictly more integer coordinates than $\mathbf{x}$. We then reach an integral optimal solution by finitely many repetitions of this procedure.

Let H be the subgraph of G induced by the set of edges $\{e \mid x_e \notin \{0,1\}\}$.

Suppose first that H contains a cycle $C = (v_1, v \ldots, v_k, v_1)$. Since G and so H is bipartite, C must be even.

Let $\varepsilon = \min_{e \in E(C)} \min\{x_e, 1 - x_e\}$. Define $\mathbf{x}'$ and $\mathbf{x}''$ as follows:

$$x'_e = \begin{cases} x_e - \varepsilon, & \text{if } e = v_i v_{i+1},\ 1 \leq i \leq k-1 \text{ and } i \text{ is odd}, \\ x_e + \varepsilon, & \text{if } e = v_i v_{i+1},\ 1 \leq i \leq k \text{ and } i \text{ is even}, \\ x_e, & \text{if } e \notin E(C). \end{cases}$$

and

$$x''_e = \begin{cases} x_e + \varepsilon, & \text{if } e = v_i v_{i+1},\ 1 \leq i \leq k-1 \text{ and } i \text{ is odd}, \\ x_e - \varepsilon, & \text{if } e = v_i v_{i+1},\ 1 \leq i \leq k-1 \text{ and } i \text{ is even}, \\ x_e, & \text{if } e \notin E(Q). \end{cases}$$

where the indices must be considered modulo k. These are two admissible solutions to (11.3). Moreover,

$$\sum_{e \in E} x_e = \frac{1}{2} \left(\sum_{e \in E} x'_e + \sum_{e \in E} x''_e \right).$$

Thus $\mathbf{x}'$ and $\mathbf{x}''$ are also optimal solutions and, by the choice of ε, one of these two solutions has more integer coordinates than $\mathbf{x}$.

Hence, we may assume that H has no cycle. Consider a longest path $P = (v_1, v_2 \ldots, v_k)$ in H. Observe if e is an edge e incident to v_1 (resp. v_k) and different from $v_1 v_2$, (resp. $v_{k-1} v_k$, then $x_e = 0$, for otherwise H would contain either a cycle or a longer path.

Defining $\varepsilon = \min_{e \in E(P)} \min\{x_e, 1 - x_e\}$ and $\mathbf{x}'$ and $\mathbf{x}''$ similarly as above, we obtain two admisible solutions to (11.3). Observe that if P is odd, then the value of $\mathbf{x}''_e$ is greater than the one of $\mathbf{x}$, which contradicts the optimality of $\mathbf{x}$. If P is even, both $\mathbf{x}'$ and $\mathbf{x}''$ are also optimal solutions and, by the choice of ε, one of these two solutions has more integer coordinates than $\mathbf{x}$. □

Remark 11.2. Observe that the proof of Proposition 11.1 easily translates into a polynomial-time algorithm for transforming an optimal solution to (11.3) into an integral optimal solution to it which is necessarily an optimal solution to (11.1). Hence, one can solve MAX MATCHING in bipartite graphs in polynomial time by computing an optimal solution to (11.3) and then modifying it into an optimal solution to (11.1).

One can show the following analogue for VERTEX COVER in bipartite graphs. (Exercise 11.4).

Proposition 11.3. *If G is bipartite, then (11.4) has an integral optimal solution, which is thus an optimal solution to (11.2). So $\nu(G) = \nu_f(G)$.*

Propositions 11.1 and 11.3, together with the Duality Theorem, now imply the following fundamental min–max theorem, due independently to König [18] and Egerváry [8].

Theorem 11.4 (KÖNIG–EGERVÁRY THEOREM)**.**
In any bipartite graph G, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover. In other words, $\mu(G) = \nu(G)$.

In fact, Propositions 11.1 and 11.3 are particular cases of a more general paradigm, called *total unimodularity*, which gives a sufficient condition for a linear programme to have an integral solution. This is discussed in Section 11.1. There are other examples for which the optimal values of the problem and its relaxation are equal. One of them is given in Subsection 11.2.1. For all these problems, solving the fractional relaxation gives a polynomial-time algorithm to solve the problem.

But very often the optimal value of the problem does not equal the optimal value of its relaxation. Moreover, for many problems like VERTEX COVER, we cannot expect any polynomial-time algorithm (unless $\mathcal{P} = \mathcal{NP}$) because they are $\mathcal{NP}$-hard. However, sometimes the optimal values of the probelm and its fractional relaxation are close to each other. In that case, solving the fractional relaxation gives an approximation of the optimal value of the problem. Moreover, one can very often derive an approximate solution of the problem from an optimal solution of the relaxation. The most common technique rounds fractional solutions to integer solutions. Some examples of deterministic and randomized roundings are given in Sections 11.2 and 11.3, respectively.

Finally, for some problems, the optimal value of the fractional relaxation is a very bad estimate for the integer linear programme. This is in particular the case for graph colouring, as shown in Section 11.4.

11.1 Total Unimodularity

We have seen that the objective objective function of a linear programme attains its optimum at one of the extreme points of the associated convex polyhedron. A polyhedron is said to be *integral*, if all its extreme points are integral. If the polyhedron $\mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0$ is integral, then every integer linear programme associated to it has the same optimal value as its fractional

relaxation. Moreover, since the Simplex Algorithm goes from extreme point to extreme point, in such a case it returns an integral solution.

Some integral polyhedra have been characterized by Hoffman and Kruskal [13]. A matrix $\mathbf{A}$ is *totally unimodular* if the determinant of each of its square submatrices is equal to 0, $+1$, or -1. Using Cramér's rule, it is easy linear algebra to show the following.

Theorem 11.5 (Hoffman and Kruskal [13])**.** *The polyhedron defined by* $\mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0$ *is integral for every integral vector* $\mathbf{b}$ *if and only if* $\mathbf{A}$ *is a totally unimodular matrix.*

In particular, if $\mathbf{A}$ *is totally unimodular and* $\mathbf{b}$ *is an integral vector, then the linear programme*

$$\textit{Maximize} \quad \mathbf{c^T x} \quad \textit{subject to :}$$

$$\begin{array}{rcl} \mathbf{Ax} & \leq & \mathbf{b} \\ \mathbf{x} & \geq & 0 \end{array}$$

has an integral optimal solution (if it has one).

Remark 11.6. This characterization requires $\mathbf{b}$ to vary. For a given vector $\mathbf{b}$ it may be true that $\{\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ is an integral polyhedron even if $\mathbf{A}$ is not totally unimodular.

Many totally unimodular matrices have been known for a long time.

Theorem 11.7 (Poincaré [22])**.** *Let* $\mathbf{A}$ *be a* $(0,-1,+1)$*-valued matrix, where each column has at most one* $+1$ *and at most one* -1*. Then* $\mathbf{A}$ *is totally unimodular.*

Proof. Let $\mathbf{B}$ be a k by k submatrix of $\mathbf{A}$. If $k = 1$, then $\det(\mathbf{B})$ is either 0, $+1$ or -1. So we may suppose that $k \geq 2$ and proceed by induction on k. If $\mathbf{B}$ has a column having at most one non-zero, then expanding the determinant along this column, we have that $\det(\mathbf{B})$ is equal to 0, $+1$, or -1, by our induction hypothesis. On the other hand, if every column of $\mathbf{B}$ has both a $+1$ and a -1, then the sum of the rows of $\mathbf{B}$ is 0 and so $\det(\mathbf{B}) = 0$. □

Seymour [23] gave a characterization of totally unimodular matrices, from which a polynomial-time algorithm to recognize such matrices follows.

In the following two subsections, we show how celebrated min-max theorems follows from total unimodularity. We then give another application of total unimodularity.

11.1.1 Matchings and vertex covers in bipartite graphs

We now show how Propositions 11.1 and 11.3 follows total unimodularity.

Let $G = (V, E)$ be a graph. The *incidence matrix* $\mathbf{A}$ of G is the matrix whose rows are indexed by V and the columns by E, and whose entries are defined by

$$a_{v,e} = \begin{cases} 1 & \text{if } v \text{ is incident to } e \\ 0 & \text{otherwise} \end{cases}$$

Hence, the matricial form of (11.3) and (11.4) are, respectively,

$$\text{Maximize } \mathbf{1}^T\mathbf{x} \text{ subject to } \mathbf{Ax} \leq \mathbf{1} \text{ and } \mathbf{x} \geq \mathbf{0}. \tag{11.5}$$

and

$$\text{Minimize } \mathbf{1}^T\mathbf{y} \text{ subject to } \mathbf{A}^T\mathbf{y} \geq \mathbf{1} \text{ and } \mathbf{y} \geq \mathbf{0}. \tag{11.6}$$

where $\mathbf{A}$ is the incidence matrix of G.

Proposition 11.8. *The incidence matrix of a bipartite graph is totally unimodular*

Proof. Left as Exercise 11.5. □

Proposition11.8 and Theorem 11.5 imply that (11.5) and (11.6) have integral optimal solutions. This is Propositions 11.1 and 11.3.

11.1.2 Flows via linear programming

The Maximum Flow Problem (7.3) can be formulated as a linear programme. Indeed writing f_{uv} instead of $f(u,v)$ and c_{uv} instead of $c(u,v)$, the problem becomes

$$\begin{aligned}
&\text{Maximize} && \sum_{v\in N^+(s)} f_{sv} \\
&\text{Subject to:} && \\
& && f_{uv} \leq c_{uv} && \text{for all } uv \in E \\
& && \sum_{u\in N^-(v)} f_{uv} - \sum_{w\in N^+(v)} f_{vw} = 0 && \text{for all } v \in V \setminus \{s,t\} \\
& && f_{uv} \geq 0 && \text{for all } uv \in E
\end{aligned}$$

Let $D = (V,E)$ be a digraph. The *incidence matrix* $\mathbf{A}$ of G is the matrix whose rows are indexed by V and the columns by E, and whose entries are defined by

$$a_{v,e} = \begin{cases} +1 & \text{if } v \text{ is the head of } e \\ -1 & \text{if } v \text{ is the tail of } e \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathbf{A}'$ be the matrix obtained from $\mathbf{A}$ by removing the rows corresponding to s and t, $\mathbf{I}$ be the $|E|$ by $|E|$ identity matrix, $\mathbf{d}$ be the indicator vector of set of arcs leaving s, and $\mathbf{c} = (c_e)_{e\in E}$. Then a possible matricial formulation of the Maximum Flow Problem is

$$\text{Maximize } \mathbf{d}^T\mathbf{f} \text{ subject to } \mathbf{If} \leq \mathbf{c} \text{ and } \mathbf{A}'\mathbf{f} = \mathbf{0} \text{ and } \mathbf{f} \geq \mathbf{0}. \tag{11.7}$$

Similarly to Theorem 11.7, one can show that the matrix $\mathbf{M} := \begin{bmatrix} \mathbf{I} \\ \mathbf{A'} \end{bmatrix}$ is unimodular (Exercise 11.6). Thus Theorem 11.5 implies that, if all capacities are integers, then the maximum value of a flow is an integer and that there exists an integral maximum flow. Moreover, as already obeserved, the Simplex Algorithm returns an integral flow. Such a flow is also returned by Ford–Fulkerson Algorithm (See Remark 7.12).

The total unimodularity also implies Ford–Fulkerson Theorem (7.7). *The maximum value of an (s,t)-flow equals the minimum capacity of an (s,t)-cut.*.

Alternative proof of Theorem 7.7. By Exercise 9.15, the dual problem of (11.7) is

$$\text{Minimize } \mathbf{c}^T\mathbf{y} \text{ subject to } \mathbf{y}+\mathbf{A'}^T\mathbf{z} \geq \mathbf{d} \text{ and } \mathbf{y} \geq \mathbf{0}, \mathbf{z} \geq \mathbf{0}. \tag{11.8}$$

Because the matrix $[\mathbf{I}, \mathbf{A'}^T]$ is totally unimodular, by Theorem 11.5, the minimum of (11.8) is attained by integral vectors $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{z}}$.

Now define $V_s = \{v \in V \setminus \{s,t\} \mid \tilde{z}_v < 0\} \cup \{s\}$ and $V_t = V \setminus V_s$. Then $C = (V_s, V_t)$ is an (s,t)-cut. We shall prove that C is a cut with capacity less or equal to the maximum flow value $v_{\max}$. By the Duality Theorem (9.10), $v_{\max} = \mathbf{c}^T\tilde{\mathbf{y}}$, and trivially $\delta(C) \geq f_{\max}$, we need to prove that $\delta(C) \leq \mathbf{c}^T\tilde{\mathbf{y}}$. But $\delta(C) = \sum_{e\in C} c(e)$ and $\mathbf{c}^T\tilde{\mathbf{y}} = \sum_{e\in E} c(e)\tilde{y}_e$. Since the $\tilde{y}_e$ are non-negative, it is sufficient to prove that $\tilde{y}(e) \geq 1$ for all $e \in C$.

Let uv be an arc of C. Recall that $\tilde{\mathbf{y}} + \mathbf{A'}^T\tilde{\mathbf{z}} \geq \mathbf{d}$. If $u \neq s$, then $\tilde{y}_e + \tilde{z}_u - \tilde{z}_v \geq 0$,with $\tilde{z}_t = 0$. But by definition of V_s, $\tilde{z}_v \geq 0$ and $\tilde{z}_u < 0$, and so $\tilde{z}_u \leq -1$, since $\tilde{\mathbf{z}}$ is integral. Hence, $\tilde{y}_e \geq 1$. If $u = s$, then $\tilde{y}_e - \tilde{z}_v \geq 1$, which again implies $\tilde{y}_e \geq 1$. □

11.1.3 Covering a strong digraph with directed cycles

Gallai–Milgram Theorem (6.14) states that every digraph D can be covered by at most $\alpha(D)$ directed paths. A natural question is to ask if a digraph could be covered by few directed cycles. In general, the answer is no as there are digraphs in which some vertices are in no cycles. But in every strong digraph, any vertex is contained in a directed cycle. In 1964, Gallai [9] conjectured an analogue of Gallai–Milgram Theorem for covering strong digraphs with directed cycles. This was proved in 2004 by Bessy and Thomassé [3].

Theorem 11.9 (Bessy and Thomassé [3])**.** *The vertex set of any non-trivial strong digraph D can be covered by $\alpha(D)$ directed cycles.*

They established it by proving a stronger result, namely a min–max theorem relating a cyclic analogue of the stability number to the minimum index of a cycle covering. Here, we present a closely related min–max theorem established by Bondy, Charbit and Sebö.

Let $D = (V,E)$ be a digraph. By a *cyclic order* of D we mean a cyclic order $O = (v_1, v_2, \ldots, v_n, v_1)$ of its vertex set V. Given such an order O, each directed cycle of D can be thought of as winding around O a certain number of times. In order to make this notion precise, we define the *length* of an arc (v_i, v_j) of D (with respect to O) to be $j-i$ if $i < j$ and $n+j-i$ if $i > j$. Informally, the length of an arc is just the length of the segment of O 'jumped' by the arc. If C is a directed cycle of D, the sum of the lengths of its arcs is a certain multiple of n. This multiple is called the *index* of C (with respect to O), and denoted $i(C)$. By extension, the *index* of a family $\mathcal{C}$ of directed cycles, denoted $i(\mathcal{C})$, is the sum of the indices of its constituent cycles.

A *weighting* of the vertices of a digraph D is a function $w : V \to \mathbb{N}$. We refer to $w(v)$ as the *weight* of vertex v. By extension, the *weight* $w(H)$ of a subgraph H of D is the sum of the weights of its vertices. If D is equipped with a cyclic order O, and if $w(C) \leq i(C)$ for every directed cycle C of D, we say that the weighting w is *index-bounded* (with respect to O). Observe that for any cycle covering $\mathcal{C}$ of D and any index-bounded weighting w,

$$i(\mathcal{C}) \geq \sum_{C\in\mathcal{C}} w(C) \geq w(D). \tag{11.9}$$

Theorem 11.10. *Let D be a digraph each of whose vertices lies in a directed cycle, and let O be a cyclic order of D. Then:*

$$\min i(\mathcal{C}) = \max w(D) \tag{11.10}$$

where the minimum is taken over all cycle coverings $\mathcal{C}$ of D and the maximum over all index-bounded weightings w of D.

In order to deduce Theorem 11.9 from Theorem 11.10, it suffices to apply it to a coherent cyclic order O of D. A cyclic order is *coherent* if every arc lies in a directed cycle of index one. Bessy and Thomassé [3] showed that every strong digraph admits a coherent cyclic order. A fast algorithm for finding coherent cyclic orders can be found in [14].

We then observe that:

- for every family $\mathcal{C}$ of directed cycles of D, we have $|\mathcal{C}| \leq i(\mathcal{C})$,
- because each vertex lies in a directed cycle and O is coherent, each vertex lies in a simple cycle, so an index-bounded weighting of D is necessarily $(0,1)$-valued,
- because each arc lies in a simple cycle, in an index-bounded weighting w no arc can join two vertices of weight one, so the support of w is a stable set, and $w(D) \leq \alpha(D)$.

Proof of Theorem 11.10. Let D be a digraph, with vertex set $V = \{v_1, \ldots, v_n\}$ and arc set $E = \{a_1, \ldots, a_m\}$. It suffices to show that equality holds in (11.9) for some cycle covering $\mathcal{C}$ and some index-bounded weighting w.

An arc (v_i, v_j) is called a *forward arc* of D if $i < j$, and a *reverse arc* if $j < i$. Consider the matrix

$$\mathbf{Q} := \begin{bmatrix} \mathbf{M} \\ \mathbf{N} \end{bmatrix}$$

where $\mathbf{M} = (m_{ij})$ is the incidence matrix of D and $\mathbf{N} = (n_{ij})$ is the $n \times m$ matrix defined by:

$$n_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is the tail of } a_j \\ 0 & \text{otherwise} \end{cases}$$

Let us show that $\mathbf{Q}$ is totally unimodular also. Consider the matrix $\widetilde{\mathbf{Q}}$ obtained from $\mathbf{Q}$ by subtracting each row of $\mathbf{N}$ from the corresponding row of $\mathbf{M}$. Each column of $\widetilde{\mathbf{Q}}$ contains one 1 and one -1, the remaining entries being 0. Thus, by Theorem 11.7, $\widetilde{\mathbf{Q}}$ is totally unimodular. Because $\widetilde{\mathbf{Q}}$ was derived from $\mathbf{Q}$ by elementary row operations, the matrix $\mathbf{Q}$ is totally unimodular too.

We now define vectors $\mathbf{b} = (b_1, \ldots, b_{2n})$ and $\mathbf{c} = (c_1, \ldots, c_m)$ as follows.

$$b_i := \begin{cases} 0 & \text{if } 1 \leq i \leq n \\ 1 & \text{otherwise} \end{cases}$$

$$c_j := \begin{cases} 1 & \text{if } a_j \text{ is a reverse arc} \\ 0 & \text{otherwise} \end{cases}$$

Before proceeding with the proof, let us make two observations:

- If $\mathbf{x} := f_C$ is the circulation associated with a directed cycle C, then $\mathbf{cx} = i(C)$, the index of C.
- If $\mathbf{Nx} \geq \mathbf{1}$, where $\mathbf{x} := \sum\{\gamma_C f_C : C \in \mathcal{C}\}$ is a linear combination of circulations associated with a family $\mathcal{C}$ of directed cycles of D, then $\mathcal{C}$ is a covering of D.

Consider the linear programme:

$$\begin{array}{rrcl} \text{Minimize} & \mathbf{c}^T\mathbf{x} & & \\ \text{subject to} & \mathbf{Qx} & \geq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0} \end{array} \tag{11.11}$$

The system of constraints $\mathbf{Qx} \geq \mathbf{b}$ is equivalent to the two systems $\mathbf{Mx} \geq \mathbf{0}$ and $\mathbf{Nx} \geq \mathbf{1}$. Because the rows of $\mathbf{M}$ sum to $\mathbf{0}$, the rows of $\mathbf{Mx}$ sum to 0, which implies that $\mathbf{Mx} = \mathbf{0}$. Thus every feasible solution to (11.11) is a non-negative circulation in D. Recall that a circulation of D is a mapping $f : E(D) \to \mathbb{R}$, such that for every vertex v $\sum_{u \in N^-(v)} f(uv) = \sum_{u \in N^+(v)} f(vu)$. Hence, a nonnegative linear combination $\sum \gamma_C f_C$ of circulations associated with directed cycles of D (Exercise 11.8). Moreover, because $\mathbf{Nx} \geq \mathbf{1}$, the cycles of positive weight in this sum form a covering of D. Conversely, every cycle covering of D yields a feasible solution to (11.11). The linear programme (11.11) is feasible because, by assumption, D has at least one cycle covering, and it is bounded because $\mathbf{c}$ is non-negative. Thus (11.11) has an optimal solution. Indeed, by Theorem 11.5, the problem (11.11) has an integral optimal solution, because $\mathbf{Q}$ is totally unimodular and the constraints are integral. This solution corresponds to a cycle covering $\mathcal{C}$ of minimum index, the optimal value being its index $i(\mathcal{C})$.

We now study the dual of (11.11):

$$\begin{array}{rrcl} \text{Maximize} & \mathbf{b}^T\mathbf{y} & & \\ \text{subject to} & \mathbf{Q}^T\mathbf{y} & \leq & \mathbf{c} \\ & \mathbf{y} & \geq & \mathbf{0} \end{array} \tag{11.12}$$

Let us write $\mathbf{y} := (z_1, \ldots, z_n, w_1, \ldots, w_n)$. Then (11.12) is the problem of maximizing $\sum_{i=1}^n w_i$ subject to the constraints:

$$z_i - z_k + w_i \leq \begin{cases} 1 & \text{if } a_j := (v_i, v_k) \text{ is a reverse arc} \\ 0 & \text{if } a_j \text{ is a forward arc} \end{cases}$$

Consider an integral optimal solution to (11.12). If we sum the above constraints over the arc set of a directed cycle C of D, we obtain the inequality

$$\sum_{v_i \in V(C)} w_i \leq i(C)$$

In other words, the function w defined by $w(v_i) := w_i$, $1 \leq i \leq n$, is an index-bounded weighting, and the optimal value is the weight $w(D)$ of D. By the Duality Theorem (Theorem 9.10), we have $i(\mathcal{C}) = w(D)$. □

11.2 Deterministic rounding

Unfortunately, most of linear programmes do not have a totally unimodular constraint matrix. However, the fractional relaxation may still be of interest. It sometimes leads to polynomial-time algorithms to solve the problem either optimally or approximately. The general idea is to first compute a solution of the fractional relaxation and then derive from this solution a solution to the original problem which is either optimal or near optimal.

11.2.1 Minimum cut

Recall that an (s,t)-cut in a graph is a bipartition (V_s,V_t) with $s \in V_s$ and $t \in V_t$. It is completely determined by the indicator vector $\mathbf{p}$ of V_t. Hence a formulation of the minimum (s,t)-cut in a graph $G=(V,E)$ is the following.

$$\text{Minimize} \quad \sum_{uv\in E} c_{uv}|p_u - p_v| \quad \text{subject to :}$$

$$\begin{array}{rcll} p_s & = & 0 & \\ p_t & = & 1 & \\ p_v & \in & \{0,1\} & \forall v \in V(G) \end{array} \tag{11.13}$$

It does not immediately appear to be an integer linear programme, but it indeed is for it is equivalent to the following.

$$\text{Minimize} \quad \sum_{uv\in E} c_{uv} q_{uv} \quad \text{subject to :}$$

$$\begin{array}{rcll} q_{uv} - p_u + p_v & \geq & 0 & \forall uv \in E \\ q_{uv} - p_v + p_u & \geq & 0 & \forall uv \in E \\ p_s & = & 0 & \\ p_t & = & 1 & \\ p_v & \in & \{0,1\} & \forall v \in V \\ q_{uv} & \in & \{0,1\} & \forall uv \in E \end{array} \tag{11.14}$$

One can check that the matrix corresponding to (11.14) is not totally unimodular (Exercise 11.10). However, solving the fractional relaxation of this problem, (where the constraint $p_v \in \{0,1\}$ is replaced by $0 \leq p_v \leq 1$), gives us the value of a minimum cut and, almost immediately, an optimal solution.

Proposition 11.11. *Let* $\mathbf{p}$ *be an optimal solution of the fractional relaxation of (11.13). Then for all* $0 \leq x < 1$*, the* $(0,1)$*-valued vector* $\mathbf{p}^x$ *defined by* $p^x_u = 0$ *if* $p_u \leq x$ *and* $p^x_u = 1$ *otherwise, is an optimal an optimal solution of (11.13) and its fractional relaxation.*

Proof. We prove it by induction on the number k of values other than 0 and 1 taken the p_v's, the result holding trivially if $k = 0$. Suppose now that $k \geq 1$. Let $0 = p_0 < p_1 < p_2 < \cdots < p_k < p_{k+1} = 1$ be the values taken by the p_v's.

Assume that $p_1 \leq x$. Let P (resp. L, R) be the set of vertices v such that $p_v = p_1$ (resp. $p_v = 0$, $p_v > p_1$).

$$\sum\{c_{vu} \mid u \in P, v \in L, uv \in E\} = \sum\{c_{vu} \mid u \in P, v \in R, uv \in E\} \tag{11.15}$$

for otherwise one of $\mathbf{p}'$ and $\mathbf{p}''$ defined by

$$p'_v = \begin{cases} p_v & \text{if } v \in V \setminus P \\ 0 & \text{if } v \in P \end{cases} \quad \text{and} \quad p''_v = \begin{cases} p_v & \text{if } v \in V \setminus P \\ p_2 & \text{if } v \in P \end{cases}$$

would contradict that $\mathbf{p}$ is an optimal solution. Furthermore, because of (11.15), $\mathbf{p}'$ (and also $\mathbf{p}''$) is an optimal solution which takes one value less than $\mathbf{p}$. Hence by induction, $\mathbf{p}^x$ is an optimal solution of (11.13) and its fractional relaxation.

A symmetrical argument yields the result if $p_k > x$. □

11.2.2 Vertex cover: 2-approximation and kernel.

If a graph is non- bipartite, then its incidence matrix is not totally unimodular (Exercise 11.5). We have also seen that for the odd cycles the fractional vertex cover number is strictly less than the vertex cover number. Moreover finding an optimal vertex cover is NP-hard, so we cannot hope to have a polynomial-time algorithm to find one. However we shall now see that there is a 2-approximate algorithm by solving FRACTIONAL VERTEX COVER and then deriving a solution to VERTEX COVER.

Theorem 11.12. FRACTIONAL VERTEX COVER *has an optimal solution which is half-integral, that is* $(0,1/2,1)$*-valued.*

Proof. Let $\mathbf{y}$ be an optimal solution of FRACTIONAL VERTEX COVER with the largest number of coordinates in $\{0,1/2,1\}$.

Suppose that $\mathbf{y}$ does not have all its coordinates in $\{0,1/2,1\}$. Set $\varepsilon = \min\{y_v, |y_v - \frac{1}{2}|, 1 - y_v \mid v \in V \text{ and } y_v \notin \{0,1/2,1\}\}$.

Consider $\mathbf{y}'$ and $\mathbf{y}''$ defined as follows:

$$y'_v = \begin{cases} y_v - \varepsilon, & \text{if } 0 < y_v < \frac{1}{2}, \\ y_v + \varepsilon, & \text{if } \frac{1}{2} < y_v < 1, \\ y_v, & \text{otherwise.} \end{cases} \quad \text{and } y''_v = \begin{cases} y_v + \varepsilon, & \text{if } 0 < y_v < \frac{1}{2}, \\ y_v - \varepsilon, & \text{if } \frac{1}{2} < y_v < 1, \\ y_v, & \text{otherwise.} \end{cases}$$

These are two admissible solutions to FRACTIONAL VERTEX COVER. Moreover, $\sum_{v\in V} y_v = \frac{1}{2}\left(\sum_{v\in V} y'_v + \sum_{v\in V} y''_v\right)$. Thus $\mathbf{y}'$ and $\mathbf{y}''$ are also optimal solutions and, by the choice of ε, one of these two solutions has more coordinates in $\{0,1/2,1\}$ than $\mathbf{y}$, a contradiction. □

Remark 11.13. Observe that the proof of Theorem 11.12 easily translates into a polynomial-time algorithm for transforming an optimal solution to FRACTIONAL VERTEX COVER into an half-integral optimal solution to it.

Corollary 11.14. *There is an algorithm which returns a* 2*-approximate solution to* VERTEX COVER *in polynomial time.*

Proof. The algorithm is very simple. We first solve FRACTIONAL VERTEX COVER and derive an half-integral optimal solution $\mathbf{y}^f$ to it. Define $\mathbf{y}$ by $y_v = 1$ if and only if $y_v^f \in \{1/2; 1\}$. Clearly, $\mathbf{y}$ is an admissible solution of VERTEX COVER. Moreover, by definition

$$\sum_{v \in V} y_v \leq 2 \sum_{v \in V} y_v^f = 2 \cdot \nu^f(G) \leq 2 \cdot \nu(G).$$

□

No better constant-factor approximation algorithm than the above one is known. The minimum vertex cover problem is APX-complete, that is, it cannot be approximated arbitrarily well unless $\mathcal{P} = \mathcal{NP}$. Using techniques from the PCP theorem, Dinur and Safra [6] proved that VERTEX COVER cannot be approximated within a factor of 1.3606 for any sufficiently large vertex degree unless $\mathcal{P} = \mathcal{NP}$. Moreover, if the unique games conjecture is true, then VERTEX COVER cannot be approximated within any constant factor better than 2 as shown by Khot and Regev [16].

Analysing in more details relations between the solutions of FRACTIONAL VERTEX COVER and those of VERTEX COVER, we can find a kernelization of the following parameterized version of the vertex cover problem.

Problem 11.15 (PARAMETERIZED VERTEX COVER)**.**
Instance: a graph G and an integer k.
Parameter: k.
Question: does G have a vertex cover of cardinality at most k ? In other words, $\nu(G) \leq k$?

In the area of parameterized algorithms, one of the major issue is to determine if a problem is FPT (for Fixed-Parameter Tractable), that is if it can be solved by an algorithm in time $f(k)P(n)$ with f an arbitrary function in k and P a polynomial in n, the number of vertices of G. One of the possible ways to do so, is to prove that there exists a $g(k)$-*kernelization* for some function $g(k)$, that is a polynomial-time algorithm that transforms any instance (G,k) of the problem into an equivalent instance (G',k') such that $|G'| \leq g(k)$ and $k' \leq k$. Once we know that a problem admits a kernelization, a second issue is to find the minimum function g for which it has a $g(k)$-kernelization.

In the remaining of this subsection, we shall prove that PARAMETERIZED VERTEX COVER has a $2k$-kernelization.

Let $\mathbf{y}^f$ be a half-integral optimal solution to FRACTIONAL VERTEX COVER. For $t \in \{0, 1/2, 1\}$, set $V_t = \{v \in V \mid y_v^f = t\}$, $G_t = G\langle V_t \rangle$.

Theorem 11.16 (Nemhauser and Trotter [21])**.** *Let $\mathbf{y}^f$ be a half-integral optimal solution to* FRACTIONAL VERTEX COVER. *Then there exists a minimum cover S of G such that:*

(a) $V_0 \cap S = \emptyset$*;*

(b) $V_1 \subseteq S$.

Proof. We present here a proof due to Khuller [17]. Let us first show that (a) implies(b). Suppose that there exists $v \in V_1 \setminus S$.

- If v has no neighbour in V_0, then one can decrease the weight y_v^f at v to obtain a fractional vertex cover of G with smaller weight, a contradiction to the minimality of $\mathbf{y}^f$.
- If v has a neighbour w in V_0, then by (a) w is not in S so vw is not covered by S, a contradiction.

Let us now prove (a).
Let S be a minimum vertex cover which has the fewest vertices in V_0. Suppose for a contradiction that $S \cap V_0 \neq \emptyset$. Observe that V_0 is a stable set because $\mathbf{y}^f$ is a fractional vertex cover. Also there is no edges between $V_0 \setminus S$ and $V_1 \setminus S$. Suppose moreover that $|V_0 \cap S| < |V_1 \setminus S|$. Then we can increase y_v^f of some tiny ε for all $v \in V_0 \cap S$ and decrease y_v of ε for all $v \in V_1 \setminus S$. This results in a fractional vertex cover of G with weight less than the one of $\mathbf{y}^f$, a contradiction.

Hence we have $|V_0 \cap S| \geq |V_1 \setminus S|$. Thus $(S \setminus V_0) \cup V_1$ is a vertex cover of G with at most as many vertices as S and no vertex in V_0. □

Corollary 11.17. PARAMETERIZED VERTEX COVER *has a* $2k$*-kernelization.*

Proof. The kernelization algorithm is the following.

Algorithm 11.1.

1. Find an optimal solution $\mathbf{y}^f$ to FRACTIONAL VERTEX COVER.
2. If the weight of $\mathbf{y}^f$ is greater than k, then return a 'No'-instance.
3. If $|V_1| = k$ and $V_{1/2} = \emptyset$, return a 'Yes'-instance.
4. Otherwise return $(G_{1/2}, k - |V_1|)$.

This algorithm clearly runs in polynomial time. In addition, by Theorem 11.16, G has a vertex cover of cardinality at most k if and only if $G_{1/2}$ has a vertex cover of cardinality at most $k - |V_1|$.

The trivial instances returned at Steps 2 or 3 being trivially of size at most $2k$, it remains to prove that the graph $G_{1/2}$ has order at most $2k$. But because of Step 2,

$$k \geq \nu^f(G) \geq \sum_{v \in V} y_v^f = \frac{1}{2}|V_{1/2}| + |V_1|$$

so $|V_{1/2}| \leq 2(k - |V_1|)$. □

11.3 Randomized rounding

11.3.1 Maximum satisfiability

There is a natural optimization version of the Boolean Satisfiability Problem (SAT) (Problem 3.5). Given a Boolean formula F in conjunctive normal form and a non-negative weight w_i

associated with each clause C_i, the objective is to find a Boolean assignment to the variables that maximizes the total weight of the satisfied clauses. This problem, called MAX SAT, is clearly NP-hard.

Johnson [15] demonstrated a simple $\frac{1}{2}$-approximation algorithm. It is based on the following simple random assignment: set each variable uniformly at random to *true* or $false$, independently from the other variables. Let $|C_i|$ denote the number of literals in clause C_i. It is easy to check that

$$p_1(C_i) = \mathbf{Pr}(C_i \text{ is satisfied}) = 1 - 2^{-|C_i|}.$$

Hence, the expected weight of clauses satisfied by this random assignment is

$$\mathbf{E}(W) = \sum_{C_i} w_i(1 - 2^{-|C_i|}) \leq \frac{1}{2}\sum_{C_i} w_i.$$

The probabilistic method specifies that there must exist a truth assignment whose weight is at least this expected value. In fact, the method of conditional probabilities (See Chapter 15 of [1]) can be applied to find such assignment deterministically in polynomial time. In the method of conditional probabilities, the value for thr ith variable is determined in the ith iteration: given the values of $x_1, \ldots, x_{i-1}$ calculate the expected weight of clauses satisfied by the probabilistic assignment, given the current assignment to $x_1, \ldots, x_{i-1}$ and the assignment $x_i = 1$. Then calculate the expected weight given the assignment to $x_1, \ldots, x_{i-1}$ and $x_i = 0$. The variable x_i is assigned the value that maximizes the conditional expectation. Since each conditional expectation can be calculated in polynomial time, the overall algorithm takes polynomial time, and as asserted above, the produced assignment has weight at least $\mathbf{E}(W)$.

Observe that the above algorithm is also a $(1 - 2^{-k})$-approximation algorithm when each clause contains at least k literals. In particular, if $k \geq 2$, the performance guarantee is at least $\frac{3}{4}$. A general $\frac{3}{4}$-approximation algorithm was proposed by Yannakakis [24]. This algorithm transforms a MAX SAT instance into an equivalent instance (in terms of approximability) which does not contain any clauses with only one literal. In conjunction with Johnson's algorithm, this leads to the performance guarantee of $3/4$. The algorithm uses maximum flow calculation in an elegant way to transform instance in which all clauses have two literals. However, the transformation becomes more complicated when general clauses are introduced. We now describe a simpler algorithm due to Goemans and Williamson with the same approximation ratio [11].

The idea is to consider two different randomized procedures for constructing the Boolean assignment, and observe that they have complementary strengths in terms of their approximation guarantees. The first one is Johnson's algorithm and thus works well when each clause has 'many' literals. The second one will be good if each clause has 'few' literals. Thus, we could run both and take the better solution; in particular, we could choose one of the two schemes uniformly at random, and the expectation of the value of the resulting solution will be the arithmetic mean of the expected values of the solutions of the two procedures.

Let us now present and analyze the second procedure. It starts with an integer linear programming formulation. For each clause C_i, let $P(i)$ denote the set of unnegated variables appearing in it, and $N(i)$ be the set of negated variables in it. For each variable j, let $x_j = 1$ if

this variable is set to *true*, and $x_j = 0$ if the variable is set to $false$. Letting $z_i \in \{0,1\}$ be the indicator for clause C_i getting satisfied, we obtain the following formulation.

$$\begin{array}{lll} \text{Maximize} & \sum_i w_i z_i & \text{subject to :} \\ z_i & \leq & \sum_{j \in P(i)} x_j + \sum_{j \in N(i)} (1 - x_j) \qquad \text{for all variable } i \end{array} \tag{11.16}$$

We first solve fractional relaxation of (11.16) obtained by relaxing each x_j and z_i to be a real in $[0,1]$ and consider a optimal solution $(\mathbf{x}^f, \mathbf{z}^f)$ of it. We interpret each x_j^f as a probability. Thus, our randomized rounding process will be, independently for each j, to set $x_j = 1$ (i.e., make variable j *true*) with probability x_j^f and $x_j = 0$ (make variable j $false$) with probability $1 - x_j^f$. One intuitive justication for this is that if x_j^f were 'high', i.e., close to 1, it may be taken as an indication by the linear programme that it is better to set variable j to *true*; similarly for the case where x_j is close to 0.

Let us lower-bound the probability of clause C_i getting satisfied. Without loss of generality, we can assume that all variables appear unnegated in C_i. Thus, we have

$$z_i^f = \min\{ \sum_{j \in P(i)} x_j^f, 1 \}.$$

It is not hard to check that $\mathbf{Pr}(C_i \text{ is satisfied}) = 1 - \prod_{j \in P(i)} (1 - x_j^f)$ is minimized when each x_j^f equals $z_i^f / |C_i|$. Thus

$$p_2(C_i) = \mathbf{Pr}(C_i \text{ is satisfied}) \geq 1 - \left(1 - \frac{z_i^f}{|C_i|}\right)^{|C_i|}.$$

For a fixed value of z_i^f, the term $1 - (1 - z_i^f / |C_i|)^{|C_i|}$ decreases monotonically as $|C_i|$ increases. This is the sense in which this scheme is complementary to Johnson's.

So, as mentioned above, suppose we choose one of the two schemes uniformly at random, in order to balance their strengths. Then,

$$\begin{array}{lll} \mathbf{Pr}(C_i \text{ is satisfied}) & = & \frac{1}{2}(p_1(C_i) + p_2(C_i)) \\ & \geq & 1 - \frac{1}{2}\left(2^{-|C_i|} + (1 - z_i^f / |C_i|)^{|C_i|}\right) \\ & \geq & \frac{3}{4} z_i^f \end{array}$$

because $z_i^f \in [0,1]$. Indeed for any fixed positive integer k, $f(k,x) = \frac{1}{2}\left(2^{-k} + (1 - x/k)^k\right) - \frac{3x}{4}$ has a non-positive derivative for $x \in [0,1]$. Thus, it suffices to show that $f(k,1) \geq 0$ for all positive integer k. We have $f(1,1) = f(2,1) = 0$. For $k \geq 3$, $2^- k \leq 1/8$ and $(1 - 1/k)^k \leq 1/e$. So $f(k,1) \geq 0$ for $k \geq 3$.

Thus the expected value of of the produced assignment is at least $3/4$ of the optimal value of the fractional relaxation of (11.16), and so at least at least $3/4$ of the optimal value of (11.16) itself.

The method of conditional probabilities may also be used to derandomized the second procedure and thus to get a deterministic $\frac{3}{4}$-approximation algorithm for MAX SAT.

11.3.2 Multiway cut

Let (G,p) be an edge-weigthed graph and let $s_1,\dots,s_k$ be k vertices called *terminals* $s_1,\dots,s_k$. An $(s_1,\dots,s_k)$-*cut* is a partition $\Pi=(V_1,\dots,V_k)$ of V such that $s_i\in V_i$ for all $1\leq i\leq k$. This notion generalizes the one of (s,t)-cut defined in Section 7.3 and Subsection 11.2.1.

Let $\Pi=(V_1,\dots,V_k)$ be a partition of G. An edge e is Π-*transversal* if its endvertices are in different parts. In other words, $e=uv$, $u\in V_i$, $v\in V_j$ and $i\neq j$. The set of transversal edges of Π is denoted $E(\Pi)$. The *weight* of a cut Π is the sum of the weights of the Π-transversal edges: $w(\Pi)=\sum_{e\in E(\Pi)} w(e)$.

The MULTIWAY CUT problem is the following:

Problem 11.18 (Multiway Cut)**.**
Instance: an edge-weighted graph (G,w) and k vertices $s_1,\dots,s_k$.
Find: an $(s_1,\dots,s_k)$-cut of minimum weight.

For $k=2$, it is the problem of finding a minimum-weight (s,t)-cut which can be solved in polynomial time as we saw in Chapter 7 and in Subsection 11.2.1.

For $k\geq 3$, the problem is $\mathcal{NP}$-hard [5]. But, it can be approximated in polynomial time. As shown by Dalhaus et al. [5], one can obtain a $2-\frac{2}{k}$-approximated solution for MULTIWAY CUT by running k times the algorithm for finding a minimum-weight (s,t)-cut. Indeed, for all i, consider the graph G_i obtained from G by identifying the s_j, $j\neq i$ in one vertex t_i and find a minimum-weight (s_i,t_i)-cut C_i in G_i. Then each C_i separates s_i from all other s_j in G. Hence the union of the $k-1$ cuts C_i with smallest weight is an $(s_1,\dots,s_k)$-cut with weight at most $2-\frac{2}{k}$ times the minimum weight of an $(s_1,\dots,s_k)$-cut. See Exercise 11.13.

A 3/2-approximation algorithm

For $1\leq i\leq k$, we denote by e_i the vector, all coordinates of which are 0 except the i^{th} which is 1. Then MULTIWAY CUT may be formulated as follows.

$$\text{Minimize} \quad \textstyle\sum_{uv\in E(G)} w(uv)d(uv) \quad \text{subject to :}$$

$$\begin{array}{rcll} d(uv) & = & \frac{1}{2}\sum_{i=1}^{k}|x_u^i-x_v^i| & \\ x_v & \in & \{e_i \mid 1\leq i\leq k\} & \text{for all } v\in V\setminus\{v_1,\dots,v_k\} \\ x_{s_i} & = & e_i & \text{for } 1\leq i\leq k \end{array} \tag{11.17}$$

Indeed a feasible solution $\mathbf{x}$ of (11.17) corresponds to the $(s_1,\dots,s_k)$-cut $(V_1,\dots,V_k)$ defined by $V_i=\{v \mid x_v=e_i\}$.

This is an integer linear programme, because Equation $(\star)$ can be replaced by following linear inequalities.

$$d(uv)=\frac{1}{2}\sum x_{uv}^i$$
$$x_{uv}^i\geq x_u^i-x_v^i \text{ for all } 1\leq i\leq k$$
$$x_{uv}^i\geq x_v^i-x_u^i \text{ for all } 1\leq i\leq k$$

Let Δ_k be the set of vectors of $\mathbb{R}^k$ whose coordinates are non-negative and sum to 1. Because a solution $\mathbf{x}$ is a vector of vectors of $\mathbb{R}^k$, for convenience we denote by $x(i)$ the i^{th} coordinate of a vector $x \in \mathbb{R}^k$. With this notation, $\Delta_k = \{x \mid \sum_{i=1}^k x(i) = 1 \text{ and } x(i) \geq 0, \forall 1 \leq i \leq k\}$. One can relax (11.17) into the following linear programme.

$$\text{Minimize} \quad \sum_{uv \in E(G)} w(uv)d(uv) \quad \text{subject to :}$$

$$\begin{array}{rcll} d(uv) & = & \frac{1}{2}\sum_{i=1}^k |x_u^i - x_v^i| & \\ x_v & \in & \Delta_k & \text{for all } v \in V \setminus \{v_1, \ldots, v_k\} \\ x_{s_i} & = & e_i & \text{for } 1 \leq i \leq k \end{array} \tag{11.18}$$

For any solution $\mathbf{x}$ of $\Delta_k^{V(G)}$, we denote by $S_{(G,w)}(\mathbf{x})$ or simply $S(\mathbf{x})$ the value $\sum_{uv \in E(G)} w(uv)d(uv)$. The idea of the approximation algorithm is to first find an optimal solution $\mathbf{x}$ of (11.18), and then to derive from this solution an $(s_1, \ldots, s_k)$-cut whose weight is at most $(3/2 - 1/k)S(\mathbf{x})$, and so at most $(3/2 - 1/k)$ times the minimum weight of an $(s_1, \ldots, s_k)$-cut.

In order to derive a multiway cut from the solution, we first transform the edge-weighted graph (G, w) and the solution $\mathbf{x}$ into an edge-weighted graph (G^*, w^*) and an admissible solution (G^*, w^*) to its associated (11.18) such that

(i) $S_{(G,w)}(\mathbf{x}) = S_{(G^*,w^*)}(\mathbf{x}^*)$, and

(ii) for all edge $uv \in E(G^*)$, the vectors x_u^* and x_v^* differ in at most 2 coordinates.

It is easy to see that such (G^*, w^*) and $\mathbf{x}^*$ can be obtained by running the following algorithm.

Algorithm 11.2.

1. $(G^*, w^*) := (G, w)$ and $\mathbf{x} := \mathbf{x}^*$.
2. While there is an edge uv such that x_u^* and x_v^* differ in $m > 2$ coordinates, do
   - Subdivide the edge uv, that is replace the edge uv by two edges uw and wv (where w is a new vertex) with weight $w^*(uw) = w^*(vw) = w^*(uv)$.
   - Choose a vector x_w^* which differs to x_u^* in exactly two coordinates and which differ to x_v^* in fewer than m coordinates.

Let us make few observations that can be easily proved.

Observation 11.19. 1) G^* is a subdivision of G and each time we subdivide an edge uv into (u, w, v), we have $d(uv) = d(uw) + d(wv)$.

2) If $\Pi^* = (V_1^*, \ldots, V_k^*)$ is an $(s_1, \ldots, s_k)$-cut in G^*, then the $(s_1, \ldots, s_k)$-cut $\Pi = (V_1, \ldots, V_k)$ in G, defined by $V_i = V_i^* \cap V(G)$ for all $1 \leq i \leq k$ has weight no greater than the one of Π^*: $w^*(\Pi^*) \leq w(\Pi)$.

We now describe a polynomial-time $(3/2-1/k)$-approximation algorithm for MULTIWAY CUT due to Calinescu et al. [4]. It first find an optimal solution of (11.18) and then deduce an admissible solution of the Multiway Cut Problem.

Let us introduce some notation. For all $1 \leq i \leq k$, let E_i be the set of edges uv of G^* such that x_u^* and x_v^* differ in the i^{th} coordinate and let $W_i^* = \sum_{e \in E_i} w^*(e)d(e)$. Finally, for $\rho \in [0,1]$ and $1 \leq i \leq 1$, let $B(s_i, \rho)$ be the set of vertices of G^* such that the i^{th} coordinate of x_v^* is at least ρ. Observe that if $\rho > 1/2$, then for $i \neq j$ we have $B(s_i,\rho) \cap B(s_j,\rho) = \emptyset$.

Algorithm 11.3 (Multiway Cut Approximation)**.**

1. Find an optimal solution $\mathbf{x}$ to (11.18).
2. Run Algorithm 11.2 to obtain (G^*, w^*) and $\mathbf{x}^*$.
3. Renumber so that $W_k^* = \max\{W_i^* \mid 1 \leq i \leq k\}$.
4. Choose ρ at random in $[0,1]$ and choose uniformly at random a permutation σ among the two $(1,2,\ldots,k-1,k)$ and $(k-1,k-2,\ldots,1,k)$.
5. For $i = 1$ to $k-1$, $V_{\sigma(i)}^* := B(s_i,\rho) \setminus \bigcup_{j<i} V_{\sigma(j)}^*$.
6. $V_k^* := V(G^*) \setminus \bigcup_{j<i} V_{\sigma(j)}^*$.
7. Return the cut $\Pi = (V_i, \ldots, V_k)$, where $V_i = V_i^* \cap V(G)$.

This algorithm is randomized but it can be derandomized.

We shall now show that Algorithm 11.3 returns a $(\frac{3}{2} - \frac{1}{k})$-approximated solution to the Multiway Cut Problem, that is $\mathbf{E}(w(\Pi)) \leq (\frac{3}{2} - \frac{1}{k}) w(\Pi_{opt})$ with Π_{opt} a minimum-weight $(s_1,\ldots,s_k)$-cut.

Since $S_{(G^*,w^*)}(\mathbf{x}^*) = S_{(G,w)}(\mathbf{x}) \leq w(\Pi_{opt})$ and $w^*(\Pi^*) \leq w(\Pi)$, it is sufficient to prove

$$\mathbf{E}(w^*(\Pi^*)) \leq \left(\frac{3}{2} - \frac{1}{k}\right) S_{(G^*,w^*)}(\mathbf{x}^*) \tag{11.19}$$

Lemma 11.20. *Let e be an edge of G^*.*

(i) If $e \in E(G^*) \setminus E_k$, then $\mathbf{Pr}(e$ is Π^*-transerval$) \leq \frac{3}{2} d(e)$.

(ii) If $e \in E_k$, then $\mathbf{Pr}(e$ is Π^*-transerval$) \leq d(e)$.

Proof. (i) Let $e = uv$ and let i and j be the two coordinates in which x_u^* and x_v^* differ.

Without loss of generality, we may assume that $x_u^*(i) < x_v^*(i)$. Since x_u^* and x_v^* differ in exactly two coordinates and the sum of the coordinates of each of these vectors equals 1, we have $x_v^*(i) - x_u^*(i) = x_u^*(j) - x_v^*(j)$. In particular, $x_u^*(j) > x_v^*(j)$.

Let us define $B = [x_v^*(j), x_u^*(j)]$ and $A = [x_u^*(i), x_v^*(i)]$ if $x_v^*(i) < x_v^*(j)$ and $A = [x_u^*(i), x_v^*(j)]$ otherwise.

Claim 11.20.1. *If $\rho \notin A \cup B$, then u and v are in the same part.*

Proof. □

Clearly, $\mathbf{Pr}(\rho \in A \cup B) = |A| + |B| \leq 2d(e)$. This would be sufficient to show $\mathbf{E}(w^*(\Pi^*)) \leq 2S_{(G^*,w^*)}(\mathbf{x}^*)$, but we want to prove that the solution is $(3/2 - 1/k)$-approximate.

Claim 11.20.2. *If $\rho \in A$ and $\sigma(j) < \sigma(i)$, then u and v are in the same part.*

Proof. □

But $\mathbf{Pr}(\rho \in A \text{ and } \sigma(j) < \sigma(i)) = \mathbf{Pr}(\rho \in A) \times \mathbf{Pr}(\sigma(j) < \sigma(i)) = d(e)/2$ because the two events '$\rho \in A$' and '$\sigma(j) < \sigma(i)$' are independent. Thus $\mathbf{Pr}(e \in E(\Pi^*)) \leq \mathbf{Pr}(\rho \in A \cup B) - \mathbf{Pr}(\rho \in A \text{ and } \sigma(j) < \sigma(i)) \leq \frac{3}{2}d(e)$. □

Finally,

$$\begin{aligned}\mathbf{E}(w^*(\Pi^*)) = \sum_{e \in E(G^*)} w^*(e)\,\mathbf{Pr}(e \in E(\Pi^*)) &\leq \sum_{e \in E_k} w^*(e)d(e) + \frac{3}{2}\sum_{e \in E(G^*)\setminus E_k} w^*(e)d(e) \\ &\leq \left(\frac{3}{2} - \frac{1}{k}\right)\sum_{e \in E(G^*)} w^*(e)d(e) = S_{(G^*,w^*)}(\mathbf{x}^*).\end{aligned}$$

The last inequality holds because $W_k^* = \max\{W_i^* \mid 1 \leq i \leq k\} \geq \frac{1}{k}\sum_{e \in E} w^*(e)d(e)$.

11.4 Graph colouring

Let $G = (V, E)$ be a graph. We denote by $\mathcal{S}(G)$, or simply $\mathcal{S}$, the set of stable sets of G and $\mathcal{S}(G, v)$, or simply $\mathcal{S}(v)$, the set of all those stable sets which include vertex v. Finding the chromatic number $\chi(G)$ of a graph G may then be formulated as an integer linear programme:

$$\begin{aligned} \text{Minimize} \quad & \sum_{S \in \mathcal{S}} x_S \\ \text{Subject to:} \quad & \\ & \sum_{S \in \mathcal{S}(v)} \geq 1 \qquad \text{for all } v \in V(G) \\ & x_S \in \{0,1\} \qquad \text{for all } S \in \mathcal{S} \end{aligned} \tag{11.20}$$

The *fractional chromatic number* of G, denoted $\chi_f(G)$, is the optimal value of the fractional relaxation of (11.20). By the Duality Theorem, the fractional chromatic number is equal to the optimal solution of

$$\begin{aligned} \text{Maximize} \quad & \sum_{v \in V(G)} y_v \\ \text{Subject to:} \quad & \\ & \sum_{v \in S} y_v \leq 1 \qquad \text{for all } S \in \mathcal{S}(G) \\ & y_v \geq 0 \qquad \text{for all } v \in V \end{aligned} \tag{11.21}$$

The feasible points of this dual linear programme are often called *fractional cliques*, the reason being that every integral feasible point (i.e., $(0,1)$-vector) identifies a clique of the graph considered. This maximization problem is henceforth called the *fractional clique problem*. Despite this pleasing result, it is still NP-hard to compute the fractional chromatic number of a graph: the number of constraints can be exponential in the size of the graph. In fact, it is even difficult to approximate the fractional chromatic number of graphs with n vertices to within a factor of $n^{1/7-\varepsilon}$ for any positive ε [2].

The fractional chromatic number may be a bad estimate for the chromatic number as the gap between those two numbers may be arbitrarily large. An interesting example of graphs for which there is a large gap between the chromatic number and the fractional chromatic number is provided by the family of *Kneser graphs*, which we now define.

For any two positive integers k and n with $k \leq n$, the *Kneser graph* $KG_{n,k}$ is the graph whose vertices are identified with the k-subsets of $\{1,\cdots,n\}$ and such that two vertices are adjacent if and only if the corresponding sets are disjoint. $KG_{n,1}$ is the complete graph K_n, which has chromatic and $KG_{5,2}$ is the Petersen graph, for which $\chi(K_{5,2}) = 3$. The graph $KG_{2k-1,k}$ has no edges, and so chromatic number 1, whereas $KG_{2k,k}$ consists of a perfect matching, and hence $\chi(KG_{2k,k}) = 2$ for all $k \geq 1$. Note that $KG(n,k)$ is vertex-transitive. Hence its fractional chromatic number is $|V(KG(n,k)|/\alpha(KG(n,k)) = n/k$. See Exercise 11.14.

Proposition 11.21. *For all $k \geq 1$ and $n \geq 2k$, the fractional chromatic number of the Kneser graph $KG_{n,k}$ is $\frac{n}{k}$.*

Theorem 11.22 (Lovász [19])**.** *For all $k \geq 1$ and $n \geq 2k-1$, the chromatic number of the Kneser graph $KG_{n,k}$ is $\chi(KG_{n,k}) = n-2k+2$.*

The proof of this theorem is one of the first graph theoretic results obtained by topological means. We refer the reader interested in such methods to the excellent monograph by Matoušek [20]. The short proof that we present next is due to Greene [12]. It uses one of the many versions of the Borsuk-Ulam Theorem, known as the Generalized Lyusternik Shnirel'man Theorem.

Theorem 11.23 (Greene [12])**.** *For any cover $A_1, A_2, \ldots, A_{n+1}$ of the n-dimensional sphere $\mathbb{S}^n$ by $n+1$ sets, each A_i open or closed, there is at least one set A_i containing a pair of antipodal points (i.e. $\{x,-x\} \in A_i$).*

Proof of Theorem 11.22. Set $d := n-2k+1$. Let $X \subset \mathbb{S}^d$ be an n-point set such that no hyperplane passing through the center of $\mathbb{S}^d$ contains more than d points of X. This condition is easily met by a set in a suitably general position, since we deal with points in $\mathbb{R}^{d+1}$ and require that no $d+1$ of them lie on a common hyperplane passing through the origin.

Let us suppose that the vertex set of $KG_{n,k}$ is the set $\binom{X}{k}$ of the k-subsets of X (in other words, we identify the elements of $\{1,\ldots,n\}$ with the points of X).

Consider a d-colouring c of $KG_{n,k}$. We shall prove that c is not proper. For $x \in \mathbb{S}^d$, let $H(x)$ be the open hemisphere $H(x)$ centered at x, that is, $H(x) = \{y \in \mathbb{S}^d \mid \langle x,y\rangle > 0\}$. We define sets $A_1,\ldots,A_d \subseteq \mathbb{S}^d$ as follows. A point $x \in \mathbb{S}^d$ is in A_i if and only if there is at least one vertex $F \in \binom{X}{k}$ of colour i contained in $H(x)$. Finally, we set $A_{d+1} = \mathbb{S}^d \setminus (A_1 \cup \cdots \cup A_d)$. Then, A_i

is an open set for each $i \in \{1,\cdots,d\}$, while A_{d+1} is closed. By Theorem 11.23, there exists $i \in \{1,\ldots,d+1\}$ and $x \in \mathbb{S}^d$ such that both x and $-x$ belong to A_i.

Suppose first that $i = d+1$. Then $H(x)$ contains at most $k-1$ points of X, and so does $H(-x)$. Therefore, the complement $\mathbb{S}^d \setminus (H(x) \cup H(-x))$, which is an "equator", (that is, the intersection of $\mathbb{S}^d$ with a hyperplane through the origin), contains at least $n-2k+2 = d+1$ points of X. This contradicts the choice of X.

Hence $i \leq d$. So we obtain two disjoint k-tuples coloured i, one in the open hemisphere $H(x)$ and one in the opposite open hemisphere $H(-x)$. Consequently, c is not proper. □

11.5 Exercises

Exercise 11.1. Formulate the Travelling Salesman Problem as an integer linear programme.

Exercise 11.2. Formulate the Hamiltonian Cycle Problem as an integer linear programme.

Exercise 11.3. Show that $\mu_f(C_{2k+1}) = k + 1/2$.

Exercise 11.4. Show that if G is bipartite, then (11.4) has an integral optimal solution.

Exercise 11.5. Let G be a graph and let $\mathbf{A}$ be its incidence matrix. Prove that $\mathbf{A}$ is totally unimodular if and only if G is bipartite.

Exercise 11.6. Prove that the matrix $[I, A'^T]$ of (11.8) is totally unimodular.

Exercise 11.7. Recall that a *stable set* in a graph is a set of pairwise non-adjacent vertices, and that an *edge cover* of a graph G is a set of edges $F \subseteq E(G)$ such that every vertex $v \in V(G)$ is incident to at least one edge of F.

Let G be a graph.
1) Formulate the problems of finding a maximum stable set and finding a minimum edge cover in G as integer linear programmes.
2) Show that the fractional relaxation of these programmes are dual.
3) Deduce the König–Rado Theorem: *In any bipartite graph without isolated vertices, the number of vertices in a maximum stable set is equal to the number of edges in a minimum edge cover.*

Exercise 11.8. Let D be a digraph.
1) Show that a non-negative circulation in D is a non-negative linear combination of circulation associated with directed cycles.
2) Show that a non-negative integral circulation in D is a non-negative integral linear combination of circulation associated with directed cycles.

Exercise 11.9 (Hoffman's circulation theorem)**.** Let $D = (V,E)$ be a directed digraph and let d and c be two weight function on the arcs of D such that $d(e) \leq c(e)$ for each arc e. Consider the problem of finding a circulation f such that $d(a) \leq f(a) \leq c(a)$ for every arc e.
1) Model this problem as a linear programmme involving the incidence matrix $\mathbf{A}$ of D.

2) Deduce from the total unimodularity of **A** Hoffman's circulation theorem: *There exists a circulation f such that $d(e) \leq f(e) \leq c(e)$ for every arc e if and only if for every subset U of V,*

$$\sum_{e \in E(V \setminus U, U)} d(e) \leq \sum_{e \in E(U, V \setminus U)} c(e).$$

Recall that $E(A,B)$ denotes the set of arcs with tail in A and head in B.

Exercise 11.10. Show that the matrix corresponding to (11.14) is not totally unimodular.

Exercise 11.11. Show that the fractional relaxation and a simple rounding yields a 2-approximate algorithm toMAXIMUM MATCHING.

Exercise 11.12. Show that every integer feasible solution **x** to the linear programme (11.5) satisfies the inequality

$$\sum_{e \in E(X)} x_e \leq \frac{1}{2}(|X| - 1)$$

for any odd subset X of V of cardinality three or more.
([7] showed that, by adding these inequalities to the set of constraints in (11.5), one obtains a linear programme every optimal solution of which is $(0,1)$-valued.)

Exercise 11.13. Let $s_1, \ldots, s_k$ be k vertices of an edge-weighted graph (G,w). For all i, consider the graph G_i obtained from G by identifying the s_j, $j \neq i$ in one vertex t_i and let C_i be a minimum-weight (s_i, t_i)-cut in G_i. Assume moreover that $w(C_k) \geq w(C_i)$, for all i.

Show that $\bigcup_{i=1}^{k-1} C_i$ is an $(s_1, \ldots, s_k)$-cut with weight at most $2 - \frac{2}{k}$ times the minimum weight of an $(s_1, \ldots, s_k)$-cut.

Exercise 11.14.
1) Show that for every graph G, $\chi_f(G) \geq \dfrac{|V(G)|}{\alpha(G)}$.
2) Show that if G is vertex-transitive, then $\chi_f(G) = \dfrac{|V(G)|}{\alpha(G)}$.
3) Decuce that $\chi_f(KG_{n,k}) = \frac{n}{k}$.

Bibliography

[1] N. Alon and J. Spencer. *The Probabilistic Method.* Second edition. Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley, New York, 2000.

[2] S. Arora and C. Lund. Hardness of approximations. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing, Boston, 1996.

[3] S. Bessy and S. Thomassé. Three min-max theorems concerning cyclic orders of strong digraphs. In *Integer Programming and Combinatorial Optimization*, 132–138. Lecture Notes in Comput. Sci., Vol. 3064, Springer, Berlin, 2004.

[4] G. Calinescu, H. Karloff et Y. Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences* 60:564–574, 2000.

[5] E. Dahlhaus, D.S. Johnson, C. H. Papadimitriou, P. D. Seymour et M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.* 23:864–894, 1994.

[6] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics* 162 (1): 439–485, 2005.

[7] J. Edmonds. Maximum matching and a polyhedron with $0,1$-vertices. *J. Res. Nat. Bur. Standards Sect. B* 69B:125–130, 1965.

[8] E. Egerváry. On combinatorial properties of matrices. *Mat. Lapok.* 38:16–28, Hungarian with German summary, 1931

[9] T. Gallai. Problem 15. In *Theory of Graphs and its Applications* (M. Fiedler, ed.), 161. Czech. Acad. Sci. Publ., 1964.

[10] M. R. Garey and D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[11] M. X. Goemans and D. P. Williamson. New 3/4-approximation algorithms for the maximum satisfiability problem. *SIAM Journal on Discrete Mathematics*, 7:656–666, 1994.

[12] J. E. Greene. A new short proof of Kneser's conjecture. *Amer. Math. Monthly*, 109(10):918–920, 2002.

[13] A. J. Hoffman and J. B. Kruskal. Integral boundary points of convex polyhedra. In *Linear Inequalities and Related Systems* (H. W. Kuhn and A. W. Tucker, eds.), Princeton University Press, pp. 223–246, 1956.

[14] S. Iwata and T. Matsuda. Finding coherent cyclic orders in strong digraphs. *Combinatorica* 28(1):83–88, 2008.

[15] D. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.* 9:256–278, 1974.

[16] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *Journal of Computer and System Sciences* 74(3): 335–349.

[17] S. Khuller. The Vertex Cover problem. *ACM SIGACT News* 33(2):31–33, 2002.

[18] D. König. Graphs and matrices. *Mat. Fiz. Lapok* 38:116–119, in Hungarian, 1931.

[19] L. Lovász. Kneser's conjecture, chromatic number, and homotopy. *J. Combin. Theory Ser. A*, 25(3):319–324, 1978.

[20] J. Matoušek. *Using the Borsuk-Ulam theorem*. Universitext. Springer-Verlag, Berlin, 2003. Lectures on topological methods in combinatorics and geometry, Written in cooperation with Anders Björner and Günter M. Ziegler.

[21] G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Math. Program.* 8:232–248, 1975.

[22] H. Poincaré. Second complément à l'analysis situs. *Proceedings of the London Mathematical Society* 32:277–308, 1900.

[23] P. D. Seymour. Decomposition of regular matroids. *Journal of Combinatorial Theory Series B* 28:305–359, 1980.

[24] M. Yannakakis. On the approximation of maximum satisfiability. *Journal of Algorithms* 17:475–502, 1994.

Chapter 12

Lagrangian Relaxation

In the first part of the course, we have succeeded to find efficient algorithms to solve several important problems such as SHORTHEST PATHS, NETWORK FLOWS... But, as we have seen, most of practical graph or network problems are NP-complete and hard to solve. In such a case, it may be interesting to solve a simplified problem to obtain approximations or bounds on the initial hardest problem. Consider the following optimization problem where $f : \mathbb{R}^n \to \mathbb{R}$ and $S \subseteq \mathbb{R}^n$:

$$\begin{array}{rl} \min & f(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in S \end{array}$$

A relaxation of the above problem has the following form:

$$\begin{array}{rl} \min & f_R(\mathbf{x}) \\ \text{subject to} & \mathbf{x} \in S_R \end{array}$$

where $f_R : \mathbb{R}^n \to \mathbb{R}$ is such that $f_R(\mathbf{x}) \leq f(\mathbf{x})$ for any $\mathbf{x} \in S$ and $S \subseteq S_R$. It is clear that the optimal solution f_R^* of the relaxation is a lower bound of the optimal solution of the initial problem. In previous section, the considered problems are such that $S = X \cap \{0,1\}^n$ where $X \subseteq \mathbb{R}^n$ (or $X \subseteq \mathbb{Q}^n$) and the fractional relaxation corresponds to consider $f_R = f$ and $S_R = X$.

A large number of these problems have an underlying network structure. The idea of the *Lagrangian Relaxation* is to try to use the underlying network structure of these problems in order to use these efficient algorithms. The Lagrangian Relaxation is a method of *decomposition*: the constraints $S = S_1 \cup S_2$ of the problems are separated into two groups, namely the "easy" constraints S_1 and the "hard" constraints S_2. The hard constraints are then removed, i.e., $S_R = S_1$ and transferred into the objective function, i.e., f_R depends on f and S_2.

Since S_R is a set of "easy" constraints, it will be possible to solve the relaxation problem. Moreover, the interest of the Lagrangian relaxation is that, in some cases, the optimal solution of the relaxed problem actually gives the optimal solution of the initial problem.

Note that this chapter is mostly inspired by [1] (Chapter 16).

We first illustrate the method on a classical example.

12.1 Constrained Shortest Paths.

Consider the network of Figure 12.1(a). Each edge ij has two attributes: c_{ij} the classical cost of the Shortest path problem and t_{ij} the time necessary to take the edge. The TIME CONSTRAINED SHORTEST PATH PROBLEM consists in finding the path of minimum cost between the source vertex (1 in the example) and the sink vertex (6 here), but restricted to the paths that require no more than T units of time to traverse.

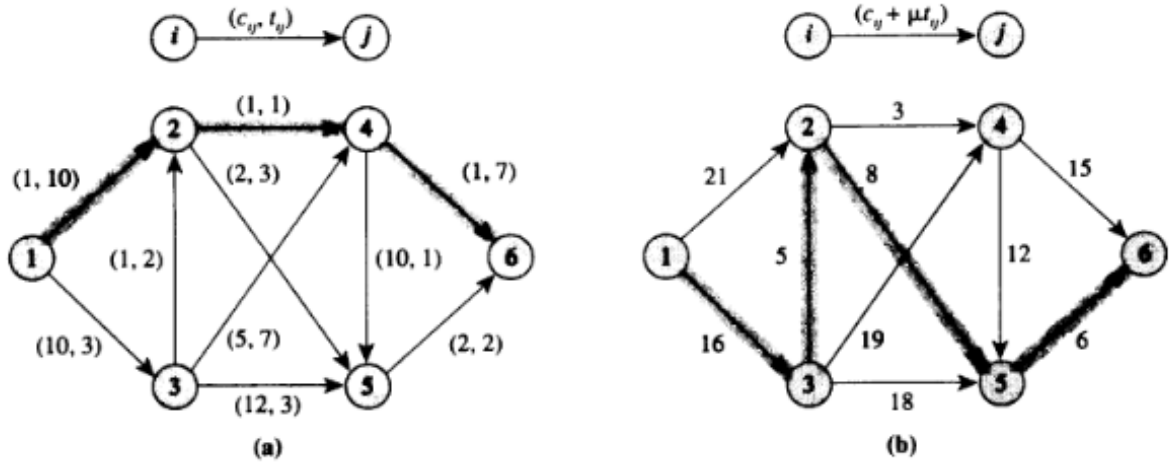


Figure 12.1:

This type of constrained shortest path application arises frequently in practice since in many contexts a company (e.g. a package delivery firm) wants to provide its services at the lowest possible cost and yet ensure a certain level of service to its customers (as embodied in the time restriction). In general, the constrained shortest path problem from node 1 to node n in the digraph $D = (V = \{1, \cdots, n\}, A)$ can be stated as the following integer programming problem

$$\begin{array}{rl} \min & \sum_{ij \in A} c_{ij} x_{ij} \\ \text{subject to} & \\ & \sum_{j;ij \in A} x_{ij} - \sum_{j:ji \in A} x_{ij} = \begin{cases} +1 & \text{if} \quad i = 1 \\ -1 & \text{if} \quad i = n \\ 0 & \text{otherwise} \end{cases} \\ & \sum_{ij \in A} t_{ij} x_{ij} \leq T \\ & x \in \{0,1\} \qquad \forall ij \in A \end{array} \tag{12.1}$$

The above problem 12.1 can clearly be decomposed into a classical (and easily solvable) shortest path problem plus an extra "time" constraint. The idea of the Lagrangian Relaxation is to include this extra constraint as a penalty in the objective function.

More precisely, let $\mu > 0$ and consider the following new problem:

$$\begin{array}{rl} \min & \sum_{ij\in A} c_{ij}x_{ij} - \mu(T - \sum_{ij\in A} t_{ij}x_{ij}) \\ \text{subject to} & \\ & \sum_{j;ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ij} = \begin{cases} +1 & \text{if} \quad i = 1 \\ -1 & \text{if} \quad i = n \\ 0 & \text{otherwise} \end{cases} \\ & x \in \{0,1\} \qquad \forall ij \in A \end{array} \tag{12.2}$$

The new problem 12.2 is equivalent to find a shortest path from node 1 to node n in D_μ, i.e., the digraph D where each arc $ij \in A$ has the modified cost $c_{ij} + \mu \cdot t_{ij}$.

$$\begin{array}{rl} \min & \sum_{ij\in A}(c_{ij} + \mu \cdot t_{ij})x_{ij} \\ \text{subject to} & \\ & \sum_{j;ij\in A} x_{ij} - \sum_{j:ji\in A} x_{ij} = \begin{cases} +1 & \text{if} \quad i = 1 \\ -1 & \text{if} \quad i = n \\ 0 & \text{otherwise} \end{cases} \\ & x \in \{0,1\} \qquad \forall ij \in A \end{array}$$

Therefore, $\mu > 0$ being fixed, Problem 12.2 can be easily solved. Let v^*_μ be the optimal solution of Problem 12.2 and let P^*_μ be a path that achieves this solution. Now, let us informally describe how we can obtain the optimal solution v^* of Problem 12.1.

First, note that any feasible solution of Problem 12.1 is a path P from node 1 to n such that $\sum_{ij\in A(P)} t_{ij} \leq T$. Therefore, because $\mu > 0$, any feasible solution P has a cost $c_\mu(P)$ in Problem 12.2 that is not larger than its cost $c(P)$ in Problem 12.1. In particular,

$$v^*_\mu \leq c_\mu(P^*) \leq c(P^*) = v^*$$

for any $\mu > 0$, where P^* is a feasible solution of 12.1 achieving the optimal value v^*. That is, the optimal solution of Problem 12.2 provides a lower bound on the optimal solution of Problem 12.1.

Let us consider an example for $T = 10$.

First, let us set $\mu = 0$. P^*_0 is the path $(1,2,4,6)$ with cost $v^*_0 = 3$. However, this path is not feasible in the initial problem. Therefore, the relaxation with $\mu = 0$ only provides that $3 \leq v^*$.

For $\mu = 1$, we get $P^*_1 = (1,2,5,6)$ and $v^*_1 = 20 - \mu \cdot T = 10$. Again, this path is not feasible in the initial problem. However, we got a better lower bound: $10 \leq v^*$.

For $\mu = 2$, two paths achieve the optimal value $v^*_2 = 15 \leq v^*$. However, on of them, $P^*_2 = (1,3,2,5,6)$ is such that $\sum_{ij\in A(P^*_2)} t_{ij} = T$. Therefore, P^*_2 is a feasible solution of Problem 12.1. Moreover, $15 = c(P^*_2) \geq v^* \geq c_2(P^*_2) = 15$. Hence, we solved Problem 12.1.

On this example, we have shown that choosing the "good" value of μ allows to obtain the optimal solution of the initial problem. In the sequels, we show how to choose a value of μ that will provide a "good" lower bound for the initial problem and we show that, in some cases, it actually leads to the optimal solution.

12.2 The Lagrangian Relaxation Technique

In this section, we formally define the Lagrangian dual problem of an optimization problem and show that the solution of the Lagrangian dual provides a lower (resp., upper) bound of the initial minimization (resp., maximazation) problem. Moreover, in the case of (convex) linear programmes, the optimal solution of the Lagrangian dual coincides with the optimal solution of the initial problem. Also, the bound obtained thanks to the Lagrangian relaxation is at least as good as the one obtained from fractional relaxation.

12.2.1 Lagrangian dual

Consider the following optimization problem:

$$\begin{array}{rl} \min & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \\ & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \in X \end{array} \tag{12.3}$$

The Lagrangian relaxation procedure uses the idea of relaxing the explicit linear constraints by bringing them into the objective function with associated vector μ called the Lagrange multiplier. We refer to the resulting problem

$$\begin{array}{rl} \min & \mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{A}\mathbf{x} - \mathbf{b}) \\ \text{subject to} & \\ & \mathbf{x} \in X \end{array} \tag{12.4}$$

as a *Lagrangian relaxation* or *Lagrangian subproblem* or the original problem 12.3, and we refer to the function

$$L(\mu) = \min\{\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{A}\mathbf{x} - \mathbf{b}) : \mathbf{x} \in X\},$$

as the Lagrangian function.

Lemma 12.1 (Lagrangian Bounding Principle)**.** *For Lagrangian multiplier μ, the value $L(\mu)$ of the Lagrangian function is a lower bound on the optimal objective function value z^* of the original optimization problem 12.3.*

Proof. Since $\mathbf{A}\mathbf{x} = \mathbf{b}$ for every feasible solution $\mathbf{x}$ of 12.3, for any Lagrangian multiplier μ, $z^* = \min\{\mathbf{c}^T\mathbf{x} : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \in X\} = \min\{\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{A}\mathbf{x} - \mathbf{b}) : \mathbf{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \in X\}$. Since removing the constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$ from the second formulation cannot lead to an increase in the value of the objective function (the value might decrease), $z^* \geq \min\{\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{A}\mathbf{x} - \mathbf{b}) : \mathbf{x} \in X\} = L(\mu)$. □

To obtain the sharpest possible lower bound, we would need to solve the following optimization problem

$$L^* = \max_{\mu} L(\mu) \tag{12.5}$$

which we refer to as the *Lagrangian Dual* problem associated with the original optimization Problem 12.3. The Lagrangian bounding principle has the following immediate implication.

12.2.2 Bounds and optimality certificates

Property 12.2 (Weak Duality)**.** *The optimal solution L^* of the Lagrangian dual problem 12.5 is a lower bound on the optimal solution z^* of Problem 12.3, i.e., $L^* \leq z^*$.*

Proof. For any Lagrangian multiplier μ and for any feasible solution $\mathbf{x}$ of Problem 12.3, we have

$$L(\mu) \leq L^* \leq \mathbf{c}^T\mathbf{x}.$$

□

Corollary 12.3 (Optimality Test)**.** *Let μ be a Lagrangian multiplier.*

If $\mathbf{x}$ is a feasible solution of Problem 12.3 satisfying $L(\mu) = \mathbf{c}^T\mathbf{x}$. Then

- *$L(\mu)$ is the optimal solution of the Lagrangian dual problem, i.e., $L^* = L(\mu)$, and*
- *$\mathbf{x}$ achieves the optimal solution of the primal Problem 12.3.*

In particular, if $L(\mu)$ is achieved by a vector $\mathbf{x}$ that is a feasible solution of Problem 12.3. Then, $L(\mu)$ is the optimal solution of the Lagrangian dual and $\mathbf{x}$ achieves the optimal solution of the primal Problem 12.3.

As indicated by the previous property, the bounding principle immediately implies one advantage of the Lagrangian relaxation approach. Indeed, in next section, we describe a method to compute the optimal solution L^* of the Lagrangian dual. Hence, the method can give us a *certificate* for guaranteeing that a given solution to the primal Problem 12.3 is optimal.

Even if $L(\mu) < \mathbf{c}^T\mathbf{x}$, having the lower bound permits us to state a bound on how far a given solution is from optimality: If $(\mathbf{c}^T\mathbf{x} - L(\mu))/L(\mu) \leq 0.05$, for example, we know that the objective function value of the feasible solution $\mathbf{x}$ is no more than 5% from optimality. This type of bound is very useful in practice. It permits us to assess the degree of sub-optimality of given solutions and it permits us to terminate our search for an optimal solution when we have a solution that we know is close enough to optimality (in objective function value) for our purposes.

Inequality constraints

In Problem 12.3, the "hard" constraints are all equalities. In practice, problems are described using inequalities. Consider the following optimization problem:

$$\begin{array}{rl} \min & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \\ & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in X \end{array} \tag{12.6}$$

In that case, we consider only Lagrangian multipliers with positive coefficients. The Lagrangian dual Problem is

$$L^* = \max_{\mu \geq 0} L(\mu) \tag{12.7}$$

In this setting, the Bounding principle and the weak duality property are still valid. However, a vector $\mathbf{x}$ may not be an optimal solution of the primal problem even if $\mathbf{x}$ is feasible for the primal problem and if $\mathbf{x}$ achieves the optimal solution of the Lagrangian dual $L^* = L(\mu)$ for some $\mu \geq 0$. The Optimality test may however be adapted in the following way:

Property 12.4. *If $L(\mu)$ is achieved by a vector $\mathbf{x}$ such that*

- $\mathbf{x}$ *is a feasible solution of Problem 12.6, and moreover*
- $\mathbf{x}$ *satisfies the complementary slackness condition $\mu^T(\mathbf{Ax}-\mathbf{b}) = 0$,*

then, $L(\mu)$ is the optimal solution of the Lagrangian dual 12.7 and $\mathbf{x}$ achieves the optimal solution of the primal Problem 12.6.

12.2.3 Linear Programming

All results presented above do not depend on the kind of considered optimization problem. More precisely, previously, the set X defining the "easy" constraints is arbitrary. We know focus on the linear programming case. More precisely, let us consider the following problem:

$$\begin{array}{rll} \min & \mathbf{c}^T\mathbf{x} & \\ \text{subject to} & & \\ & \mathbf{Ax} = \mathbf{b} & \mathbf{A} \in \mathbb{R}^{m_1 \times n} \\ & \mathbf{Dx} \geq \mathbf{q} & \mathbf{D} \in \mathbb{R}^{m_2 \times n} \\ & \mathbf{x} \in \mathbb{R}^n \quad \mathbf{x} \geq 0 & \end{array} \tag{12.8}$$

Recall that the corresponding dual problem is (see Section 9.3.2):

$$\begin{array}{rll} \max & (\mathbf{b}^T\mathbf{q}^T)\mathbf{y} & \\ \text{subject to} & & \\ & (\mathbf{A}^T\mathbf{D}^T)\mathbf{y} \geq \mathbf{c} & \\ & \mathbf{y} \in \mathbb{R}^{m_1+m_2} \quad \mathbf{y} \geq 0 & \end{array} \tag{12.9}$$

For any vector μ, we set the Lagrangian function as

$$L(\mu) = \min\{\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{Ax}-\mathbf{b}) : \mathbf{Dx} \geq \mathbf{q}, \mathbf{x} \geq 0\}. \tag{12.10}$$

Theorem 12.5. *Let (P) be any linear programme such as Problem 12.8. The optimal value $L^* = \max_\mu L(\mu)$ of the Lagrangian dual of (P) coincides with the optimal value z^* of (P).*

Proof. Let x^* be a vector achieving the optimal solution z^* of Problem 12.8 and let $y^* = \binom{\pi^*}{\gamma^*}$ be an optimal solution of the dual 12.9 where $\pi^* \in \mathbb{R}^{m_1}$ is associated to constraints $\mathbf{A}$ and $\gamma^* \in \mathbb{R}^{m_2}$ is associated to constraints $\mathbf{D}$.

Since y^* is a feasible solution for 12.9, we have that $\mathbf{A}^T\pi^* + \mathbf{D}^T\gamma^* - \mathbf{c} \geq 0$. Moreover, by the complementary slackness conditions (Theorem 9.11):

- $(\mathbf{A}^T\pi^* + \mathbf{D}^T\gamma^* - \mathbf{c})x^* = 0$

- $y^*(\binom{\mathbf{A}}{\mathbf{D}}x^* - \binom{b}{q}) = \gamma^*(\mathbf{D}x^* - \mathbf{q}) = 0$

Let us set $\mu = -\pi^*$, we have $L(-\pi^*) = \min\{\mathbf{c}^T\mathbf{x} - (\pi^*)^T(\mathbf{A}\mathbf{x} - \mathbf{b}) : \mathbf{D}\mathbf{x} \geq \mathbf{q}, \mathbf{x} \geq 0\}$. That is, any vector achieves $L(-\pi^*)$ if and only if it achieves the optimal solution of the following linear programme:

$$\min\{(\mathbf{c}^T - (\pi^*)^T\mathbf{A})\mathbf{x} : \mathbf{D}\mathbf{x} \geq \mathbf{q}, \mathbf{x} \geq 0\} \tag{12.11}$$

Moreover, the corresponding dual problem is

$$\max\{\mathbf{q}^T\mathbf{y} \ : \ \mathbf{D}^T\mathbf{y} \leq \mathbf{c} - \mathbf{A}^T\pi^*, \mathbf{y} \geq 0\}. \tag{12.12}$$

Therefore, the complementary slackness conditions implies that if there is a vector $\mathbf{x}'$ feasible for Problem 12.11 and a vector $\mathbf{y}'$ feasible for Problem 12.12 such that

- $\mathbf{y}'(\mathbf{D}\mathbf{x}' - \mathbf{q}) = 0$, and
- $(\mathbf{D}^T\mathbf{y}' - \mathbf{c} + \mathbf{A}^T\pi^*)\mathbf{x}' = 0$

then $\mathbf{x}'$ achieves the optimal solution of Problem 12.11.

Since we have seen that, setting $x' = x^*$ and $y' = \gamma'$ satisfies the complementary slackness conditions, this implies that x^* achieves the optimal solution of Problem 12.11.

Thus, $L(-\pi^*) = \mathbf{c}^T\mathbf{x}^* + \mu^T(\mathbf{A}\mathbf{x}^* - \mathbf{b}) = \mathbf{c}^T\mathbf{x}^* = z^*$. The result follows Corollary 12.3. □

Theorem 12.6. *Let X be a finite set in $\mathbb{R}^n$ and let $\mathcal{H}(X)$ its convex hull. Then, the Lagrangian dual of* $\min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in X\}$ *has the same optimal solution as* $\min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in \mathcal{H}(X)\}$.

Proof. Let $L(\mu) = \min\{\mathbf{c}^T\mathbf{x} + \mu(\mathbf{A}\mathbf{x} - \mathbf{b}) \ : \ \mathbf{x} \in X\}$. It is equivalent to $L(\mu) = \min\{\mathbf{c}^T\mathbf{x} + \mu(\mathbf{A}\mathbf{x} - \mathbf{b}) \ : \ \mathbf{x} \in \mathcal{H}(X)\}$ because the solutions of the second formulation are reached at some vertices of the polytope $\mathcal{H}(X)$ and that any vertex of $\mathcal{H}(X)$ belongs to X.

Therefore, the solution of the Lagrangian dual of the initial problem equals the solution of the Lagrangian dual of $\min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in \mathcal{H}(X)\}$.

The convex hull of a finite number of points can be defined as the intersection of a finite family of half-spaces, i.e., by a finite number of inequalities. Therefore, applying previous theorem to $\min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in \mathcal{H}(X)\}$, we get that its optimal solution and the one of its Lagrangian dual coincide. □

Theorem 12.7. *Let (ILP) be an integer linear programme. Then the bound achieved by a Lagrangian relaxation of (ILP) is at least as good as the result of its fractional relaxation.*

Proof. Consider the integer linear programme $(ILP) = \min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in X \cap \mathbb{Z}^n\}$ where X is convex since it is defined by linear inequalities. By previous theorem, its Lagrangian relaxation has the same solution as $(LR) = \min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in \mathcal{H}(X \cap \mathbb{Z}^n)\}$.

Since the convex hull $H = \mathcal{H}(X \cap \mathbb{Z}^n)$ of $X \cap \mathbb{Z}^n$ is such that $H \subseteq X$, we get that the solution of (LR) is not better than the one of $(LP) = \min\{\mathbf{c}^T\mathbf{x} \ : \ \mathbf{A}\mathbf{x} = \mathbf{b}, \ \mathbf{x} \in X\}$, the fractional relaxation of (ILP). That is, $(LP) \leq (LR) \leq (ILP)$. □

12.2.4 Solving the Lagrangian dual

In this section, we describe a method to approximate the solution of the Lagrangian dual

$$L^* = \max_{\mu} L(\mu) \tag{12.13}$$

of the following Lagrangian function relaxes the constraints $\mathbf{Ax} = \mathbf{b}$:

$$L(\mu) = \min\{\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{Ax} - \mathbf{b}) : \mathbf{x} \in X\},$$

Recall that the principle of the Lagrangian relaxation is to include the "hard" constraints into the objective function. In other words, optimizing a linear function over X is assumed to be "easy". Therefore, μ being fixed, we can compute the value of $L(\mu)$ and a corresponding vector $\mathbf{x} \in X$.

Note that the Lagrangian function is the lower envelope of the set of the hyperplanes $\mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{Ax} - \mathbf{b})$ for $\mathbf{x} \in X$. Therefore, $L(\mu)$ is a concave function. Say differently, it is equivalent to solve the following linear programme:

$$\begin{array}{ll} \max & w \\ \text{subject to} & \\ & w \leq \mathbf{c}^T\mathbf{x} + \mu^T(\mathbf{Ax} - \mathbf{b}) \quad \mathbf{x} \in X,\ \mu \in \mathbb{R}^n \end{array}$$

Since generally the number of constraints of such a programme is exponential, we use a gradient descent method to compute a value as close as desired to the optimal solution of the Lagrangian dual.

More precisely, given a concave function $f : \mathbb{R}^n \to \mathbb{R}$, a vector $\mathbf{g}$ is a *subgradient* of f at $\mathbf{x}$ if, for any $\mathbf{y} \in \mathbb{R}^n$, $f(\mathbf{y}) \leq f(\mathbf{x}) + \mathbf{p}^T(\mathbf{y} - \mathbf{x})$. The function f is *differentiable* in $\mathbf{x}$ if and only if f admits a unique subgradient in $\mathbf{x}$.

if $L(\mu)$ was differentiable we would use the gradient descent method to converge toward the optimal value. However, in our case, $L(\mu)$ is not differentiable everywhere since it is a piecewise linear function. So the following subgradient method is commonly used. The method computes a sequence of $(\mu_k)_{k\in\mathbb{N}}$ such that $L(\mu_k)$ converges to the optimal solution.

More precisely,

Algorithm 12.1 (Subgradient method)**.**

Initially, let $\mu_0 \in \mathbb{R}^n$, $k = 0$;

1. Given $\mu_k \in \mathbb{R}^n$, compute $L(\mu_k)$ and a vector $\mathbf{x}_k \in X$ where it is achieved;
2. Choose a subgradient $\mathbf{g}_k = \mathbf{Ax}_k - \mathbf{b}$ of the function L at μ_k;
3. If $\mathbf{g}_k = 0$, then stop, the optimal solution is $L(\mu_k)$
4. Compute $\mu_{k+1} = \mu_k + \theta_k^T.\mathbf{g}_k$ where θ_k is the stepsize at this step.
5. Increment k and go to step 2.

In practice, the heuristic to decide the stepsize is $\theta_k = \frac{UB - L(\mu_k)}{||\mathbf{A}\mathbf{x}_k - \mathbf{b}||^2}$ where UB is an upper bound on the optimal solution we want to compute.

In the case when, in the initial programme, the constraints that we relax were $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, then the method must be slightly modified such that $\mu_k \geq 0$ for any $k \geq 0$. For this purpose, at step 4., the i^{th} coefficient of μ_{k+1} is taken as the maximum between 0 and the i^{th} coefficient of $\mu_k + \theta_k^T . \mathbf{g}_k$.

In all cases, the number of steps depends on the desired accuracy of the result.

12.3 Applications

Bibliography

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, 1993.

Chapter 13

Primal-Dual Algorithms

The primal-dual methodology is a very general tool to derive algorithms for a large set of problems. It leads to algorithms providing exact solutions for a variety of polynomial problems and good approximate solutions for NP-complete problems. Most of the famous algorithms of combinatorial optimization can be interpreted as primal-dual algorithms, including Dijkstra's shortest path algorithm, Ford and Fulkerson's network flow algorithm, Edmonds' non bipartite matching algorithm and Kuhn's assignment algorithm.

In this chapter, we examine a specific combinatorial optimization problem, *the hitting set problem*. We build step by step a primal-dual method to solve this problem. While doing so, we have the stupefaction and pleasure to rediscover several classical algorithms such as Dijsktra's mentioned algorithm and Kruskal's method to obtain a spanning tree which appear as a simple and elegant application of the primal-dual methodology. This chapter was strongly inspired by [1, 2].

13.1 The Principle of Primal-Dual Algorithms in Few Words

The principle of Primal-Dual algorithms follows from Theorem 13.1. The theorem states that, if x is a feasible solution of the Primal and y is a solution of the Dual, there are optimal solution if and only if they satisfy both

1. the Primal Complementary Slackness (PCS), i.e., either $x_j = 0$ or $\sum_{i=1}^{m} a_{ij} y_i = c_j$.

2. the Dual Complementary Slackness (DCS), i.e., either $y_i = 0$ or $\sum_{j=1}^{n} a_{ij} x_j = b_i$.

The algorithm starts with a feasible solution of the Dual, y. Note that usually, it is a lot easier to find a feasible solution the Dual than to find an optimal solution of the Primal. We then take $x = 0$. In this way, the PCS is satisfied. Note that x may not be a feasible solution of the primal and that the DCS may not be satisfied.

We want now to increase the value of the x_i's so that x becomes feasible. The PCS tells us that to do so, we need to have tight dual constraints to do so. The idea is then to raise the value of the y_i's either simultaneously or one by one. Whenever a dual constraint becomes tight, we

freeze the values of the corresponding y's. We can then raise the value of the corresponding x. We repeat the process till all the y_i's are frozen.

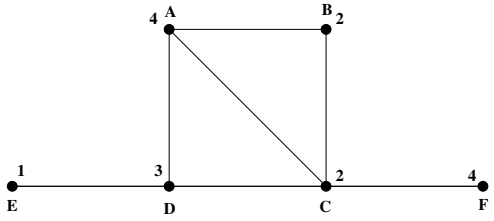
Theorem 13.1 (Complementary Slackness)**.** *Let $(x_1,\ldots,x_n)$ be a feasible solution of Problem 9.5 and $(y_1,\ldots,y_m)$ be a feasible solution of Problem 9.6. These are optimal solutions if and only if*

$$\sum_{i=1}^{m} a_{ij}y_i = c_j, \quad \text{or } x_j = 0, \quad \text{or both} \quad \text{for all } 1 \le j \le n, \text{ and}$$
$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad \text{or } y_i = 0, \quad \text{or both} \quad \text{for all } 1 \le i \le m.$$

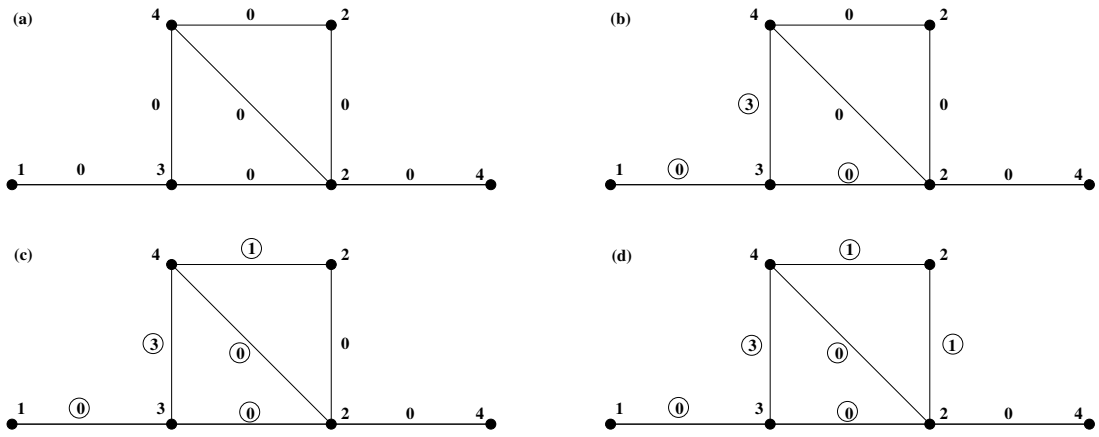
13.2 A First Example: Primal-Dual Algorithm for Vertex Cover

1. Start with $x = 0$ and $y = 0$. 2. Pick any edge e for which y_e is not frozen yet. 3. Raise the value of y_e until some vertex constraint v goes tight. 4. Freeze all y_e's for edges incident to v. Raise x_v to 1. 5. Repeat until all y_e's are frozen.

Let us see an example of how this algorithm works on an instance of the vertex cover problem. We consider the following weighted graph: We start by assigning $y_e = 0$ for edges $e \in E$



(step (a)). At each step, an edge is picked for which y_e is not frozen yet. The value of y_e is raised until the constraint of one vertex incident to e goes tight. All the edges incident to that vertex are then frozen and the value of x_v is raised to 1 (that is the vertex v is chosen in the solution of the primal problem). When all the y_e's are frozen, the algorithm terminates. In steps (a)-(d), the



chosen edges were respectively AD, AB, BC, CF and the tight vertices D, A, B and C. In this case, the value of the primal is 11 (the sum of the weights of the tight vertices) and the value of the dual is 6 (the sum of the values of the chosen edges). We have $Val_p(x) \leq Val_d(y)$ and so we get the 2-approximation for our instance of vertex cover.

Let us now prove that the algorithm always give a 2-approximation on any instance.

Lemma 13.2. *When the algorithms ends, x is a feasible solution for the primal and y is a feasible solution for the dual.*

Proof. That y is a feasible solution is obvious since at every step we make sure that y is a feasible solution, so it is feasible at the last step when the algorithm ends. To see that x is a feasible solution we proceed by contradiction. Let us suppose it is not a feasible solution. Then there is a constraint $x_u + x_v \geq 1$ which is violated. That means both x_u and x_v are zero and thus it must be possible to raise the value of the edge between them since neither of them has a tight bound. This is a contradiction with the fact that all y_e's are frozen. □

Lemma 13.3. *x and y satisfy PCS.*

Proof. By construction of the algorithm. At every step, we make sure that x and y satisfy PCS. □

Definition 13.4. Let x and y be feasible solutions to the primal and the dual respectively. Then we say that x and y $\alpha-$approximate DCS if $\forall j, (y_j \neq 0) \rightarrow (\sum_i A_{ij} x_i \leq \alpha b_j)$

Lemma 13.5. *x and y satisfy 2-approximate DCS.*

Proof. This follows from the fact that x_v is either 0 or 1 for any v so $x_u + x_v \leq 2$ for any $e = (u, v)$. □

The next lemma shows us why we would want an $\alpha-$approximation for DCS.

Lemma 13.6. *Suppose x and y satisfy PCS are feasible for Primal and Dual respectively and $\alpha-$approximate DCS then $Val_P(x) \leq \alpha Val_D(y)$.*

Proof. To prove this, we only need to write out the sums. We have $Val_P(x) = \sum_i c_i x_i$. Now since we know that x and y satisfy PCS, we have that $\sum_i c_i x_i = \sum_i (\sum_j A_{ij} y_j) x_i$. By reordering the summation we get $\sum_i (\sum_j A_{ij} x_i) y_j \leq \sum_j \alpha b_j y_j = \alpha \sum_j b_j y_j = \alpha VAl_D(y)$ where the $\leq$ follows from the α-approximate DCS. □

The last two lemmas then directly yield the desired result which is that our algorithm is a 2-approximation for Vertex Cover.

13.3 The Journey of the Hitting Set Problem

The *hitting set problem* is defined as follows: Given subsets $T_1, \ldots, T_p$ of a ground set E and given a nonnegative cost c_e for every element $e \in E$, find a minimum-cost subset $A \subseteq E$ such that $A \cap T_i \neq$ for every $i = 1, \ldots, p$ (i.e., A "hits" every T_i).

This problem can be seen as a generalisation of several well-known combinatorial problems. The *undirected $s-t$ shortest path problem* with nonnegative lengths can be formulated as a hitting set problem by noticing that any $s-t$ path must intersect every $s-t$ cut $\delta(S)$, where $\delta(S) = e = (i, j) \in E : i \in S, j \notin S$. So we can let E be the edge set of the undirected graph $G = (V, E)$; c_e be the length of the edge e; and $T_1, \ldots, T_p$ be the collection of all $s-t$ cuts, i.e. $T_i = \delta(S_i)$ where S_i rens over all sets containing s but not t. Observe that the feasible solutions consist of subgraphs in which s and t and connected; only minimal solutions (i.e., solutions for which no edge can be removed without destroying feasability) will correspond to $s-t$ paths. We leave to the interested reader the task to show that the *minimum spanning tree problem*, *the vertex cover problem* and the *minimum-cost arborescence problem* can be similarly expressed hitting set problems (Exercise 13.1).

Cost of the Solution. The cost of the solution is $c(A) = \sum_{e \in A} c_e$. As e was added to A only if the corresponding dual constraint was tight, we can rewrite the cost as $\sum_{e \in A} \sum_{i: e \in T_i} y_i$. By exchanging the two summations, we obtain

$$c(A) = \sum_{i=1}^{p} |A \cap T_i| y_i.$$

Since y is a dual feasible solution, its value $\sum_{i=1}^{p} y_i$ is a lower bound on the optimum value z_{OPT} of the hitting set problem. If we can guarantee that

$$|A \cap T_i| \leq \alpha \text{ whenever } y_i > 0$$

then this would immediately imply that $c(A) \leq \alpha z_{OPT}$, i.e., the algorithm is an α-approximation algorithm. In particular, if α can be guaranteed to be 1, then the solution given by the algorithm must certainly be optimal. Note that we can obtain here an α-approximation algorithm of the hitting set problem by choosing α to be the largest cardinality of any set T_i: $\alpha = \max_{i=1}^{p} |T_i|$.

Implementation and Efficiency Issues.

13.4 Exercises

Exercise 13.1. Express the following problems as a hitting set problem.

1. *Minimum spanning tree problem.*
2. The *Vertex cover problem* is the problem of finding a minimum (cardinality or cost) set of vertices in an undirected graph such that every edge has at least one endpoint in the set.
3. *Minimum-cost arborescence problem.* We are given a directed graph $G = (V, E)$ with non-negative arc costs and a special root vertex r. The *minimum-cost arborescence problem* is to find a spanning tree directed out of r of minimum cost.

Bibliography

[1] Goemans, M.X. and Williamson, D.P. The primal-dual method for approximation algorithms and its application to network design problems In Approximation algorithms for NP-hard problems, pages 144-191, 1997.

[2] Shuchi Chawla Primal-Dual Algorithms Lecture notes, `http://pages.cs.wisc.edu/~shuchi/courses/787-F07/scribe-notes/lecture15.pdf`, 2007.

Part III

Lecture Notes (2nd term)

Chapter 14

Radio channel assignment and (weighted) colouring

14.1 Modelling the channel assignment problem

We may think of the radio channel assignment problem as the final stage in the design of a cellular radio communications system. The general idea of such a system is that many low-powered transmitters (base stations) each serve the customers in their local cell, and thus the same radio channel can be used simultaneously in many different cells, as long as these cells are sufficiently well separated. Since the radio spectrum is a finite resource which is heavily in demand, we want to assign the channels to the transmitters carefully in order to take maximum advantage of this re-use possibility.

Suppose then that transmitters are located at various sites in a geographical region, perhaps a city, with power levels set. Engineers often aim to spread the transmitters out to form roughly a part of a triangular lattice, since it gives the best "coverage", that is, it minimises the maximum distance to a transmitter. Sometimes the transmitters may be spread out very differently, for example along a major road. The service region is divided into cells around each transmitter. A cell around transmitter v may be seen as the potential receiver sites which are closer to v than to any other transmitter, at least in the case when each transmitter as the same power. When such transmitters are spread out like part of the triangular lattice, the cells are hexagonal.

For each cell, there is an estimate of the (peak period) expected demand. Using these demand estimates and some (simple) queuing model, an appropriate number $p(v)$ of channels is chosen for each transmitter v. (Note that we are considering a static model : there is interest also in dynamic models, where the demand levels change over the time, and the focus is on the method for re-assigning channels.) The aim is to find an assignment of $p(v)$ channels to each transmitter v, such that the corresponding interference is acceptable, and the span of channel used is minimised.

So, when will interference be acceptable. Typically a "protection ratio" θ is set, depending on engineering considerations involving the selectivity of the equipment used and the width of the channel. We say that the interference arising from some channel assignment is acceptable if the signal-to-interference ratio is at least θ at each potential receiver site.

We assume that the power received does not depend on the frequency used (which is realistic since the range of frequencies involved is usually small). Another simplifying assumption that seems reasonable from the physics of interference is that only the difference between two channels matters. Typically the smaller the difference the greater the interference. But this is not always the case as there may for example be "intermodulation products", in particular at transmitters on the same site.

Consider a pair of transmitters u and v, and suppose that they transmit on channels differing by c. If there is a potential receiver in the cell around u such that the ratio of the received power from u to that from v is less than the protection ratio θ, then we make c a "forbidden distance" for u and v; similarly with u and v interchanged. Hence for each pair (u,v) of distinct transmitters, we have a set T_{uv} of forbidden differences $|i-j|$ for channels i at u and channels j at v. Similarly, for each vertex v, we have a set T_{vv} of forbidden differences between two channels at v.

We form the corresponding *interference* or *constraint* graph G. It has a vertex for each transmitter and distinct vertices u and v are adjacent if T_{uv} is non-empty. In particular, at each vertex v, there is a loop of length $l(vv) = T_{vv}$. It is often convenient to think of the problem as being specified by the graph G with a set T_e for each e of G, where always $0 \in T_e$. In the case, when the interference increases with the proximity between two transmitters then T_e is of the form $\{0,\ldots,l(e)-1\}$. Hence we may consider that each edge is associated a non-negative integer length $l(e)$ for each edge e.

An assignment is a mapping $\phi : V \to \mathcal{P}(\{1,\ldots,t\})$ such that $|\phi(v)| = p(v)$ for each $v \in V(G)$. It is *feasible* if the following conditions hold:

- (i) for any two distinct adjacent vertices u and v and each $i \in \phi(u)$ and each $j \in \phi(v)$ we have $|i-j| \geq l(uv)$.

- (ii) for each $v \in V(G)$ and any two distincts integers i and j in $\phi(v)$, we have $|i-j| \geq l(vv)$.

The *span* of the problem, $\text{span}(G,l,p)$, is the least integer t such that there is a feasible assignment. (Some authors call $t-1$ the span.)

We want to determine or approximate the span, and find corresponding assignments.

Examples

1. If G is a triangle with each edge of length 3 and the demand of each vertex is 1, then the span is 7.

2. If G is a 4-cycle with each edge length 3 and the demand of each vertex is 1, then the span is 4.

3. Let G be the 5-cycle plus the loops on all the vertices, such that every edge has length 1 except the loops which have length 2. If each vertex has demand 2, then the span is 5.

14.2 General results

In this section we give various results, some introductory, about the span in the channel assignment problem. We restrict our attention here to the case of unit demands. In this case, the co-site constraints (=loops) are irrelevant and we can see an assignment as a mapping $\phi : V \to \{1,\ldots,t\}$ which is *feasible* if $|\phi(u)-\phi(v)| \geq l(uv)$ for every edge uv. For convenience, we denote $\text{span}(G,l,\mathbf{1})$ by $\text{span}(G,l)$ where $\mathbf{1}$ the appropriate all 1's function.

Note that a general channel assignment problem can always be transformed into a unit demand problem by blowing up each vertex v into a clique of $p(v)$ vertices with edge lengths equal to $l(vv)$.

14.2.1 All equal edge lengths

Observe that $\text{span}(G,\mathbf{1})$ equals the chromatic number $\chi(G)$. Note also that, with any positive edge lengths, the least *number* of integers required is just $\chi(G)$, but it is the span that is of interest to us.

When the edge lengths are all the same, we are almost back to colouring. Let $\mathbf{k}$ denote the appropriate all k's function.

Proposition 14.1. *If each edge length is k then*

$$\text{span}(G,\mathbf{k}) = k(\chi(G)-1)+1.$$

Proof. Observe that the span is at most the right hand side, since we could always first properly colour G with $\chi(G)$ colours and then assign a channel to each colour, using channels $1, k+1,\ldots,k(\chi(G)-1)+1$.

Now let us show that the span is at least the right hand side. Let t be the span, and consider a feasible assignment ϕ using channels $0,1,\ldots,t-1$ which uses as few as possible channels which are not multiples of k. Then in fact ϕ must use only multiples of k, for otherwise the least channel not a multiple of k could be pushed down to the nearest multiple of k, giving a contradiction. But now if we let $c(v) = \phi(v)/k$ we obtain a proper colouring of G. So $\chi(G) \leq (t-1)/k+1$, which is the desired inequality. □

14.2.2 Lower bound for the span

It follows from Proposition 14.1 that if G is the complete graph K_n and all edge lengths are at least k then $\text{span}(G,l) \geq k(n-1)+1$. This result can be extended as follows. A path P is *hamiltonian* in G if its goes through all the vertices, i.e. $V(P) = V(G)$.

Proposition 14.2. *If G is complete, then*

$$\text{span}(G,l) \geq \text{hp}(G,l)+1,$$

where $\text{hp}(G,l)$ *is the minimum length of a hamiltonian path.*

Proof. Given a feasible assignment ϕ, list the vertices as $v_1, \ldots, v_n$ so that $\phi(v_1) \leq \phi(v_2) \leq \cdots \leq \phi(v_n)$. $P = v_1 v_2 \ldots v_n$ is a hamiltonian path in G and

$$\phi(v_n) - \phi(v_1) = \sum_{i=1}^{n-1} \phi(v_{i+1}) - \phi(v_i) \geq \sum_{i=1}^{n-1} l(v_i v_{i+1}),$$

which is the length of P. Since $\text{span}(G,l) = \phi(v_n) - \phi(v_1) + 1$, we get the result. □

This last result has the drawback that it is NP-hard to calculate $\text{hp}(G,l)$, but there are good lower bounds which may be efficiently calculated, for example the minimum length of a spanning tree. Observe that Proposition 14.2 is tight if the edge lengths satisfy the triangle inequality, but we should not expect this to hold for minimum channel separations.

14.2.3 Sequential assignment methods

Suppose that we want to colour the vertices of a graph with colours $1, 2, \ldots$ and we have a given ordering on the vertices. Let us consider two variants of the greedy colouring algorithm. In the "one-pass" method, we run through the vertices in order and always assign the smallest available colour. In the "many-passes" method, we run through the vertices assigning colour 1 whenever possible, then repeat with colour 2 and so on. Both methods yield exactly the same colouring, and show that $\chi(G) \leq \Delta(G) + 1$ colours.

Let us now consider the channel assignment problem (G,l). Define the *weighted degree* of a vertex v by $d_{(G,l)}(v) = \sum_{uv \in E} l(uv)$ and define the *maximum weighted degree* by $\Delta(G,l) = \max_{v \in V(G)} d_{(G,l)}(v)$.

Example: Let G be the 4-cycle C_4, with vertices v_1, v_2, v_3, v_4 and edge lengths $l(v_1v_2) = 1$ and $l(v_2v_3) = l(v_3v_4) = l(v_4v_1) = 2$. Note that $\Delta(G,l) = 4$. The one-pass method assigns channels 1,2,4,6 to the vertices v_1, v_2, v_3, v_4 respectively, with span 6. The many-passes method assigns channel 1 to vertices v_1 and v_3, channel 2 to none of the vertices, and channel 3 to vertices v_2 and v_4, with span 3. In fact the many passes method always uses at most the channels $1, \ldots, \Delta(G,l) + 1$.

Theorem 14.3 (McDiarmid [7])**.**

$$\text{span}(G,l) \leq \Delta(G,l) + 1.$$

Proof. In order to show that the many-passes method needs a span of at most the above size, suppose that it is about to assign channel c to vertex v. Let A be the set of neighbours u of v to which it has already been assigned a channel $\phi(u)$. For each channel $j \in \{1, \ldots, c-1\}$, there must be a vertex $u \in A$ with $\phi(u) \leq j$ and $\phi(u) + l(uv) \geq j + 1$. Hence the intervals $\{\phi(u), \ldots, \phi(u) + l(uv) - 1\}$ for $u \in A$ cover $\{1, \ldots, c-1\}$. Thus

$$c - 1 \leq \sum_{u \in A} l(uv) \leq d_{(G,l)}(v) \leq \Delta(G,l).$$

This completes the proof. □

14.3 Computing the span

We noted earlier that the special case when all lengths are 1 is essentially the graph colouring problem. Since graph colouring is NP-hard – see for example [3]–, we cannot expect an easy ride. In fact, the general problem seems to be harder than graph colouring.

14.3.1 Bipartite graphs and odd cycles

Let G be a graph and l an edge-length. Let $L(G,l) = \max\{l(xy)+1 \mid xy \in E(G)\}$. For any (G,l), clearly $\text{span}(G,l) \geq L(G,l)$.

This inequality is tight for bipartite graphs.

Proposition 14.4. *If G is a bipartite graph, then* $\text{span}(G,l) = L(G,l)$*, for any l.*

Proof. If we set $\phi(x) = 1$ for x in one part of the bipartition and $\phi(x) = L(G,l)$ for x in the other part, then we obtain a feasible assignment with span $L(G,l)$. □

This proposition implies that the channel assignment problem for bipartite graphs is easy. After bipartite graphs the next thing to consider is odd cycles. Here again it is easy to determine the span.

Proposition 14.5. *If G is an odd cycle then* $\text{span}(G,l) = \max\{L(G,l), M(G,l)\}$*, where* $M(G,l) = \min\{l(uv)+l(vw)+1 \mid uv, vw \in E(G)\}$.

Proof. Since G is an odd cycle, in any feasible assignment ϕ there exist edges uv and vw of G such that $\phi(u) \leq \phi(v) \leq \phi(w)$. Then $|\phi(w) - \phi(u)| \geq l(uv) + l(vw)$ and so the span of (G,l) is at least $M(G,l)$. Hence it is at least $\max\{L(G,l), M(G,l)\}$.

On the other hand, let us choose two edges uv and vw in G with $l(uv)+l(vw) = M(G,l)-1$. Form an even cycle G' by deleting v and adding the edge uw. Consider the length function l' on $E(G')$ which satisfies $l'(uw) = l(uv)+l(vw)$ and agrees with l elsewhere. Since G' is bipartite, (G',l') admits an optimal feasible assignment ϕ with span $L(G',l') = \max\{L(G,l), M(G,l)\}$. Furthermore, $|\phi(u) - \phi(w)| \geq l(uw) \geq l(uv)+l(vw)$. Hence one can choose $\phi(v)$ between $\phi(u)$ and $\phi(w)$ so that ϕ is a feasible assignment of (G,l). □

Let us call a graph 1-*nearly bipartite* if by deleting at most one vertex we may obtain a bipartite graph. It is of course easy to recognise if a graph is 1-nearly bipartite, by simply deleting each vertex in turn. It is also easy to determine the chromatic number of a 1-nearly bipartite graph G, as it is at most 3. However, it is NP-hard to determine $\text{span}(G,l)$, even if we restrict the edge lengths to be 1 or 2, see [8].

14.3.2 A general exponential algorithm

Theorem 14.6 (McDiarmid [7])**.** *Given (G,l) with maximum edge-length m, we can compute* $\text{span}(G,l)$ *in $O^*((2m+1)^n)$ steps.*

Proof. Let us describe the method. Let V denote the set of vertices of G. For each $S \subset V$, let $N_i(S) = \{v \in V \setminus S \mid \exists$ an edge uv with $u \in S$ and $l(uv) \geq i\}$. For each nested family $A \supseteq B_1 \supseteq \cdots \supseteq B_{m-1}$ of m subsets of V and each non-negative integer t, let $F(A;B_1,\ldots,B_{m-1};t)$ be the set of all feasible assignments $\phi : A \to \{1,\ldots,t\}$ for the subproblem on A such that $\phi(v) \leq t-i$ whenever $v \in B_i$, for each $i = 1,\ldots,m-1$. Let $f(A;B_1,\ldots,B_{m-1})$ be the least t such that $F(A;B_1,\ldots,B_{m-1};t)$ is non-empty. Thus the span is $f(V;\emptyset,\ldots,\emptyset)$. By definition, if $A = \emptyset$ then $F = \{\emptyset\}$ and $f = 0$.

Claim 14.6.1. *For each non-empty $A \subset V$*

$$f(A;B_1,\ldots,B_{m-1}) = 1 + \min_S f(A \setminus S;B'_1,\ldots,B'_{m-1}),$$

where S runs over all stable subsets of $A \setminus B_1$; $B'_{i-1} = B_i \cup (A \cap N_i(S))$ for each $i = 2,\ldots,m-1$; and $B'_{m-1} = A \cap N_m(S)$.
(Note that $A \setminus S \supseteq B'_1 \supseteq \cdots \supseteq B'_{m-1}$, as required for the domain of f.)

The method to calculate the span is brutal: we use the claim to tabulate all the values $f(A;B_1,\ldots,B_{m-1})$ in increasing order of the size of A. For a given set A of size a, there are m^a points in the domain of f: for each point we have m choices for the smallest element of the nested family that contains a . The additional time to compute f for a given point with set A of size a is at most $O^*(2^a)$ since they are at most 2^a (stable) sets in A. Hence the total time taken is at most $O^*\left(\sum_{a=0}^{n} \binom{n}{a} m^a 2^a\right) = O^*\left((2m+1)^n\right)$. It remains only to prove the claim.

Proof Claim 14.6.1. We show first that the left side is at most the right. Let S be a stable subset of $A \setminus B_1$, and let $f(A \setminus S;B'_1,\ldots,B'_{m-1}) = t-1$. We want to show that $f(A;B_1,\ldots,B_{m-1}) \leq t$. Let $\phi \in F(A \setminus S;B'_1,\ldots,B'_{m-1})$ and extend ϕ to $\hat{\phi} : A \to \{1,\ldots,t\}$ by setting $\hat{\phi}(v) = \phi(v)$ for each $v \in A \setminus S$ and $\hat{\phi}(v) = t$ for each $v \in S$.

We must check that $\hat{\phi} \in F(A;B_1,\ldots,B_{m-1})$. Let uv be an edge with $u \in S$ and $v \in A \setminus S$. Thus $\hat{\phi}(u) = t$ and $\hat{\phi}(v) \leq t-1$. If $l(uv) = i \in \{2,\ldots,m\}$, then $v \in N_i(S) \subseteq B'_{i-1}$, and so $\hat{\phi}(v) = \phi(v) \leq (t-1)-(i-1) = t-i$. Thus in each case $\hat{\phi}(u) - \hat{\phi}(v) \geq l(uv)$. Since S is stable and ϕ is feasible for the subproblem on $A \setminus S$, it now follows easily that $\hat{\phi}$ is feasible for the subproblem on A. If $v \in B_1$ then $\hat{\phi}(v) = \phi(v) \leq t-1$ since $S \subseteq A \setminus B_1$; and if $v \in B_i$ for some $i \in \{2,\ldots,m-1\}$ then $v \in B'_{i-1}$ and so $\hat{\phi}(v) = \phi(v) \leq t-i$ by our choice of ϕ. Hence $\hat{\phi} \in F(A;B_1,\ldots,B_{m-1})$.

Conversely, let us show that the right side is at most the left. Let $f(A;B_1,\ldots,B_{m-1}) = t$ and let $\phi \in F(A;B_1,\ldots,B_{m-1})$. Let S be the stable set $\phi^{-1}(t)$. Then S must be non-empty by the minimality of t, and $S \subseteq A \setminus B_1$ since $\phi(v) \leq t-1$ for each $v \in B_1$. If $t = 1$ then $S = A$ and the result holds, so let us assume that $t \geq 2$. Define $\phi' : A \setminus S \to \{1,\ldots,t-1\}$ by setting $\phi'(v) = \phi(v)$ for each $v \in A \setminus S$.

Let us check that $\phi' \in F(A \setminus S;B'_1,\ldots,B'_{m-1})$. Clearly ϕ' is feasible for the subproblem on $A \setminus S$. Let $i \in \{2,\ldots,m-1\}$ and let $v \in B'_{i-1} = B_i \cup (A \cap N_i(S))$. If $v \in B_i$ then $\phi'(v) = \phi(v) \leq t-i = (t-1)-(i-1)$ by the condition on ϕ, and if $v \in N_i(S)$ then the same inequality holds, since S is non-empty and ϕ is feasible for A. Finally, if $v \in B'_{m-1} = A \cap N_m(S)$, then as before $\phi'(v) = \phi(v) \leq t-m = (t-1)-(m-1)$. Thus $\phi' \in F(A \setminus S;B'_1,\ldots,B'_{m-1})$. □

The proof of Claim 14.6.1 completes the proof of Theorem 14.6. □

An Integer Programme model

The following integer programme (IP) gives a simple reformulation of the channel assignment model, though other formulations may be better suited to computation for particular types of problem, see also [10].

Choose an upper limit f_{max}, and let $F = \{1, \ldots, f_{max}\}$ be the set of available channels. We introduce a binary variable $y_{u,i}$ for each transmitter u and channel i : setting $y_{u,i} = 1$ will correspond to assigning channel i as one of the channels at transmitters u. Then $\text{span}(G, l, p)$ is given by the following integer programme.

Minimise z subject to :

$$\begin{array}{rcll} z & \geq & j \times y_{v,j} & \forall v \in V(G), j \in F \\ \sum_{j \in F} y_{v,j} & = & p(v) & \forall v \in V(G) \\ y_{u,i} + y_{v,j} & \leq & 1 & \forall u, v \in V(G) \text{ and } i, j \in F \text{ such that } (u,i) \neq (v,j) \text{ and } |i-j| < l(uv) \\ y_{v,j} & \in & \{0,1\} & \forall v \in V(G), j \in F \end{array}$$

To see that this IP formulation is correct, consider an optimal assignment $\phi : V \to F$. Set $y_{v,j} = 1$ if $j \in \phi(v)$ and $y_{v,j} = 0$ otherwise; and set z to be the maximum channel used. It is easy to see that this gives a feasible solution to the IP, wiht $z = \text{span}(G, l, p)$. Conversely, given a feasible solution to the IP with value t, we may obtain in a similar way a feasible assignment ϕ.

14.4 Channel assignment in the plane

It is natural to specialise the channel assignment problem to the case where the transmitter sites are located in the plane, and the minimum channel separation for a pair of sites depends on the distance between them.

In this section, we will consider only co-channel interference, which corresponds to each minimum channel separation being 0 (if there is no edge) or 1 (at the same site or between adjacent sites). Hence we have to determine $\text{span}(G, \mathbf{1}, p)$. Then we are left with a colouring problem of a weighted graph. A *weighted graph* is a pair (G, p), where G is a graph and p a weight function on the vertex set of G. A *t-colouring* of a weighted graph (G, p) is a mapping $C : V(G) \to \mathcal{P}(\{1, \ldots, t\})$ such that for every vertex $v \in V(G)$, $|C(v)| = p(v)$ and for all edge $uv \in E(G)$, $C(u) \cap C(v) = \emptyset$. The *chromatic number* of a weighted graph (G, p), denoted $\chi(G, p)$, is the least integer t such that (G, p) admits a t-colouring. This is a natural generalisation of the chromatic number of a graph since $\chi(G, \mathbf{1}) = \chi(G)$. Moreover, $\chi(G, p) = \text{span}(G, \mathbf{1}, p)$.

The *clique number* of a weighted graph (G, p), denoted $\omega(G, p)$, is the maximum weight of a clique, that is $\max\{p(C) \mid C \text{ clique of } G\}$, where $p(C) = \sum_{v \in C} p(v)$.

Generalising Proposition 8.3, for any weighted graph (G, p), we have

$$\chi(G, p) \geq \omega(G, p).$$

14.4.1 Disk graphs

Suppose that we are given a threshold distance d_0, such that interference will be acceptable as long as no channel is re-used at sites less than distance d_0 apart. Given a set of V points in the plane and given $d_0 > 0$, let $G(V, d_0)$ denote the graph with vertex set V in which distinct vertices u and v are adjacent whenever the euclidean distance between them is less than d_0. Equivalently, we may centre an open disk of diameter d_0 at each point v, and then two vertices are adjacent when their disks meet. Such a graph is called a *unit disk* (or *proximity*) graph.

Observe that if G is a unit disk graph, the graph G_p obtained from the weighted graph (G, p) by replacing each vertex v by a complete graph of size $p(v)$ is also a unit disk graph. Hence our basic version of the channel assignment problem is equivalent to colouring unit disk graphs. The following result shows that the clique and chromatic numbers of such graphs are not too far apart.

Proposition 14.7 (Clark, Colbourn and Johnson [2])**.** *For a unit disk graph G,*

$$\chi(G) \leq 3\omega(G) - 2.$$

Proof. In a realisation of G with diameter d_0, consider the "bottom left" point v. All its neighbours lie within an angle of less than 180 degrees at v. Thus we can cover all the neighbours with three sectors, each with radius less than d_0 and angle less than 60 degrees. But the points in each sector together with v form a clique, and so the degree of v is at most $3(\omega(G) - 1)$. It follows that the degeneracy of G is at most $3\omega(G) - 3$. The result follows from Proposition 8.7. □

It would be nice to improve this result: perhaps the factor 3 could be replaced by $3/2$? It is shown in [1] that it is NP-hard to recognise unit disk graphs. Many problems are NP-hard for unit disk graphs, even given a realisation in the plane, see [2]: for example finding $\chi(G)$ or $\alpha(G)$. However there is a polynomial time algorithm to find $\omega(G)$, given a realisation in the plane.

Theorem 14.8 (Clark, Colbourn and Johnson [2])**.** *Given a realisation of a unit disk graph G, there is a polynomial time algorithm to find* $\omega(G)$.

Proof. Suppose we are given a representation with diameter 1. The algorithm relies on the following fact. For every clique K, there are points u and v in K at euclidean distance $d < 1$ such that all points of K is contained in the region R_{uv} of points in the plane within distance at most d of both u and v. See Figure 14.1.

Hence we consider in turn each edge uv of G and the corresponding region R_{uv} and find a maximum clique in R_{uv}. The line uv cuts R_{uv} into two halves: let A and B be the sets of points in the two halves. It is easy to verify that A and B are cliques.

Let H be the bipartite graph with bipartition (A, B) where $a \in A$ and $b \in B$ are adjacent if they are at euclidean distance at least 1. Then we obtain a maximum clique within L by forming $(A \cup B) \setminus C$, where C is a minimum cover in H. □

If the transmitters can have different powers, we are led to consider disk graphs, which are defined as for unit disk graphs except that the diameters may be different.

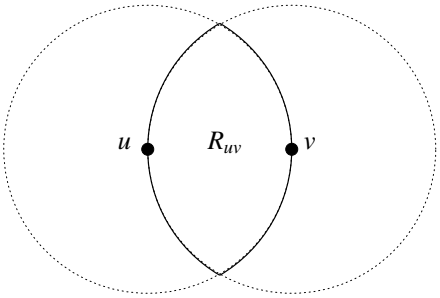


Figure 14.1: The region R_{uv}.

Proposition 14.9. *For a disk graph G,*

$$\chi(G) \leq 6\omega(G) - 5.$$

Proof. Consider a vertex v with disk of smallest diameter, and proceed as in the proof of Proposition 14.7 to show that the degeneracy is at most $6(\omega(G)-1)$. □

14.4.2 Triangular lattice

The triangular lattice TL crops up naturally in radio channel assignment. It is sensible to aim to spread the transmitters out to form roughly a part of a triangular lattice, with hexagonal cells, since that will give the best "coverage", that is, for a given number of transmitters in a given area this pattern minimises the maximum distance to a transmitter.

The triangular lattice graph TL may be described as follows. The vertices are all integer linear combinations $a\mathbf{e_1} + b\mathbf{e_2}$ of the two vectors $\mathbf{e_1} = (1,0)$ and $\mathbf{e_2} = (\frac{1}{2}, \frac{\sqrt{3}}{2})$. Thus we may identify the vertices with the pairs (a,b) of integers. Two vertices are adjacent when the Euclidean distance between them is 1. Therefore, each vertex $x = (a,b)$ has six neighbours: its *left neighbour* $(a-1,b)$, its *right neighbour* $(a+1,b)$, its *leftup neighbour* $(a-1,b+1)$, its *rightup neighbour* $(a,b+1)$, its *leftdown neighbour* $(a,b-1)$ and its *rightdown neighbour* $(a+1,b-1)$. See Figure 14.2.

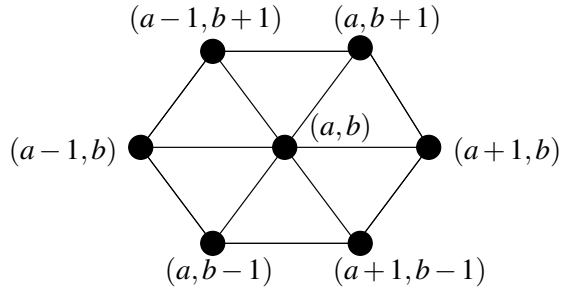


Figure 14.2: The vertex (a,b) and its six neighbours.

A *hexagonal graph* is an induced subgraph of the triangular lattice.

The triangular lattice has a unique (up to colours permutations) 3-colouring c_T , defined by $c_T((a,b)) = a - b \bmod 3$. See Figure 14.3. It follows immediately that for any weighted

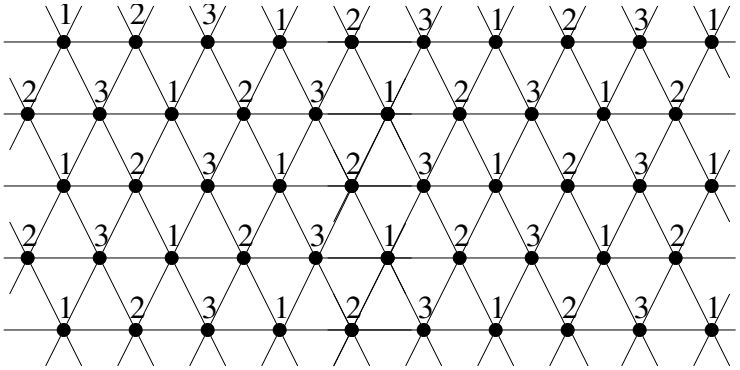


Figure 14.3: The unique 3-colouring of the triangular lattice.

hexagonal graph (G,p),

$$\chi(G,p) \leq 3\max\{p(v) \mid v \in V(G)\} \leq 3\omega(G,p).$$

In the rest of this section, we will improve this upper bound. It is fairly easy to improve it for bipartite graphs.

Lemma 14.10. *Let (G,p) be a weighted bipartite graph. Then $\chi(G,p) = \omega(G,p)$.*

Proof. Let (A,B) be a bipartition of G. Let v be a vertex of G. Assign it the colour set $\{1,2,\ldots,p(v)\}$ if it is in A and the colour set $\{\omega(G,p) - p(v) + 1,\ldots,\omega(G,p)\}$ if it is in B. This give us an optimal colouring of (G,p). Indeed, if uv is an edge with $u \in A$ and $v \in B$ then $p(u) + p(v) \leq \omega(G,p)$. □

Remark 14.11. This lemma gives us a polynomial time algorithm to compute the chromatic number of a bipartite weighted graph. Indeed, calculating $\omega(G,p)$ is easy because it is the maximum of $\max\{p(v) \mid v \in V(G)\}$ and $\max\{p(u) + p(v) \mid uv \in E(G)\}$ since a clique has size at most 2 in a bipartite graph.

McDiarmid and Reed [**?**] showed that it is NP-complete to decide whether the chromatic number of a weighted hexagonal graph is 3 or 4. Hence, there is no polynomial time algorithm for finding the chromatic number of weighted hexagonal graphs (unless P=NP). Therefore, one has to find approximate algorithms. The better known so far has approximation ratio $4/3$ and is based on the following result:

Theorem 14.12 (McDiarmid and Reed [**?**])**.** *For any weigthed hexagonal graph G,*

$$\chi(G,p) \leq \frac{4\omega(G,p)+1}{3}.$$

Proof. The weighted colouring of (G,p) with $\frac{4\omega_p(G)+1}{3}$ colours is given by the following algorithm : We first calculate the 3-colouring c_T of G. We then compute $\omega(G,p)$ and set $k = \lfloor \frac{\omega(G,p)+1}{3} \rfloor$. The algorithm proceeds in two stages.

In the first stage, we use $3k$ colours (i,j) for $i = 1,2,3$ and $j = 1,\ldots,k$. For each vertex v, we compute the value $m(v)$ which is the maximum of $p(u)$ over its neighbours u such that $c_T(u) = c_T(v)+1 \bmod 3$. (These are its right, leftup and leftdown neighbours if they are present.) If v has no such neighbours then $m(v) = 0$. Let $r(v) = \min\{p(v)-k, k-m(v)\}$. We assign to v the colours $(c_T(v),1),\ldots,(c_T(v),\min\{k,p(v)\})$. Moreover, if $r(v) > 0$ we assign to v the colours $(c_T(v)+1, k-r(v)+1),\ldots,(c_T(v)+1,k)$. By definition of $r(v)$ those colours are not assigned to the neighbours of v, so the colouring is proper.

Now let U be the set of vertices whose demand is not yet fulfilled after the first stage. Then $v \in U$ if and only if $p(v) > \max\{k, 2k-m(v)\}$, and in this case, the number of colours that remain to be assigned to v is $p'(v) = p(v) - \max\{k, 2k-m(v)\}$.

Let H be the graph induced by the vertices of U. H is triangle-free because $p(v) \geq k+1$ for every vertex $v \in U$. Hence $\omega(H) \leq 2$. In addition, for all vertex $v \in U$, $p'(v) \leq p(v)+m(v)-2k \leq \omega(G,p)-2k$ and for any two neighbours u and v in H, $p'(u)+p'(v) \leq \omega(G,p)-2k$. Hence $\omega(H,p') \leq \omega(G,p)-2k$.

We shall prove that H is acyclic and thus bipartite. Hence by Lemma 14.10, one can assign $p'(v)$ colours to every vertex v of U using $\omega(H,p')$ colours. Hence in total, we have used at most

$$3k + \omega(H,p') \leq \omega(G,p) + k \leq \frac{4\omega(G,p)+1}{3}$$

colours.

Remains to prove that H is acyclic. Observe that the left-most vertex in a cycle in a hexagonal graph has at least two neighbours to its right, that is among its rightup neighbour , its right neighbour and its rightdown neighbour. Hence it is sufficient to prove that in H a vertex v has at most one neighbour to its right Since H is triangle-free, it suffices to prove that v, its rightup neighbour x and its rightdown neighbour z cannot be all three in U. Suppose for a contradiction that these three vertices are in U. Set $s = \min\{p(x),p(z)\}$. Then $s \geq k+1$. Furthermore, if $m(v) > 0$ and u is the neighbour of v in which $m(v)$ is attained, then v, u and either x or z form a triangle, so $p(v)+m(v)+s \leq \omega(G,p)$. Notice that this inequality is also true if $m(v) = 0$ because $p(v)+s \leq \omega(G,p)$ for v is adjacent to x and z. Thus we have

$$1 \leq p'(v) \leq p(v)+m(v)-2k \leq \omega(G,p)-s-2k \leq \omega(G,p)-3k-1 \leq 0$$

which is a contradiction. □

A distributed algorithm which guarantees the $\frac{4}{3}\omega(G,p)$ bound is reported by Narayanan and Schende [11]. However, one expects to have approximate algorithms with ratios better than $4/3$. In particular, Reed and McDiarmid conjecture that, for big weights, the ratio may be decreased to almost $9/8$.

Conjecture 14.13 (McDiarmid and Reed [**?**])**.** There exists a constant c such that for any weigthed hexagonal graph (G,p),

$$\chi(G,p) \leq \frac{9}{8}\omega(G,p) + c.$$

Note that the ratio $9/8$ in the above conjecture is the best possible. Indeed, consider a 9-cycle C_9 with constant weight k. A colour can be assigned to at most 4 vertices, so $\chi(C_9,\mathbf{k}) \geq \frac{9k}{4}$. Clearly, $\omega(C_9,\mathbf{k}) = 2k$. So $\chi(C_9,k) \geq \frac{9}{8}\omega(C_9,\mathbf{k})$. An evidence for this conjecture has been given by Havet [5], who proved that if a hexagonal graph G is triangle-free (i.e. has no K_3) then $\chi(G,p) \leq \frac{7}{6}\omega(G,p)+5$. See also [12] for an alternative proof and [6] for a distributed algorithm for colouring triangle-free hexagonal graphs with $\frac{5}{4}\omega(G,p)+3$ colours.

Exercises

Exercise 14.1. Let (G,l) be a unit demand instance of the channel assignment problem and m a non-negative integer. A subset U of vertices is *m-assignable* if $\text{span}(G\langle U\rangle,l) \leq m$. Let α^m denote the maximum size of an m-assignable set. Show that

$$\text{span}(G,l) \geq \frac{m \times |V(G)|}{\alpha^m} - m - 1.$$

Exercise 14.2. Let G be a bipartite graph and p a demand function. Let $p_{\max} = \max\{p(v) \mid v \in V(G)\}$ be the maximum demand.

1. Let l be the length defined by $l(uv) = 1$ for all $uv \in E(G)$ and $l(vv) = 3$ for all $v \in V(G)$.
   a. Show that $3p_{\max} - 2 \leq \text{span}(G,l,p) \leq 3p_{\max} - 1$.
   b. Show that $\text{span}(G,l,p) \leq 3p_{\max} - 1$ if and only if there are two adjacent vertices with demand $p_{\max}$.
2. Let l be the length defined by $l(uv) = 1$ for all $uv \in E(G)$ and $l(vv) = 2$ for all $v \in V(G)$.
   a. Show that $2p_{\max} - 1 \leq \text{span}(G,l,p) \leq 2p_{\max}$.
   b. A path in G is *critical* if it has an even number of vertices, the endvertices have demand $p_{\max}$ and the internal vertices have demand $< p_{\max}$.
   Show that $\text{span}(G,l,p) = 2p_{\max} - 1$ if and only if for every critical path P, $p(P) \leq (|V(P)|-1)p_{\max} - (|V(P)|-2)/2$. (Gerke [4]).

Exercise 14.3. A *bicolouring* of a graph G is a colouring of $(G,\mathbf{2})$, where $\mathbf{2}$ is the appropriate all 2's function. A t-bicolouring is a bicolouring in $\{1,\ldots,t\}$.

1 Let $P = (x_0,x_1,\ldots,x_m)$ be a path of length $m \geq 4$. Show that for any 2-subsets c_0 and c_m of $\{1,\ldots,5\}$, there is a 5-bicolouring such that $C(x_0) = c_0$ and $C(x_m) = c_m$.

2 a) Show that every triangle-free hexagonal graph is 5-bicolourable.
b) Deduce that for any weighted hexagonal graph (G,p), $\chi(G,p) \leq \frac{5}{4}\omega(G,p)+3$.(Havet [5])

Bibliography

[1] H. Breu and D. G. Kirkpatrick. Unit disk graph recognition is NP-hard. *Comput. Geom.* 9:3–24, 1998.

[2] B. N. Clark, C. J. Colbourn and D. S. Johnson. Unit disk graphs *Discrete Math.* 86:165–177, 1990.

[3] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* A Series of Books in the Mathematical Sciences. W. H. Freeman and Co., San Francisco, Calif., 1979.

[4] S. Gerke. Colouring weighted bipartite graphs with a co-site constraint. *Discrete Mathematics* 224:125–138, 2000.

[5] F. Havet. Channel assignment and multicolouring of the induced subgraphs of the triangular lattice. *Discrete Mathematics* 233:219–231, 2001.

[6] F. Havet and J. Zerovnik. Finding a five bicolouring of a triangle-free subgraph of the triangular lattice. *Discrete Mathematics* 244:103–108, 2002.

[7] C. McDiarmid. On the span in channel assignment problems: bounds, computing and counting. *Discrete Math.* 266:387–397, 2003.

[8] C. McDiarmid and B. Reed. Channel assignment and weighted coloring. *Networks* 36:114–117, 2000.

[9] C. McDiarmid and B. Reed. Channel assignment on graphs of bounded treewidth. *Discrete Math.* 273(1-3):183-192, 2003.

[10] R. A. Murphy, M. D. Pardalos and M. G.C. Resende. Frequency assignment problems. Chapter in *Handbook of Combinatorial Optimization* Vol. 4, (D.-Z. Dhu and P.M. Pardalos, editors), 1999.

[11] L. Narayanan and S. Schende. Static Frequency Assignment in Cellular Networks. In *Sirocco 97, (Proceedings of the 4th international Colloqium on structural information and communication complexity, Ascona, Switzerland)*, D. Krizanc and P. Wildmayer (eds.), Carleton Scientific pp. 215–227, 1997.

[12] K. S. Sudeep and S. Vishwanathan. A technique for multicoloring triangle-free hexagonal graphs. *Discrete Mathematics* 300(1-3), 256–259, 2005.