Massachusetts Institute of Technology
Michel X. Goemans

18.433: Combinatorial Optimization
February 4th, 2015

1. Lecture notes on bipartite matching

Matching problems are among the fundamental problems in combinatorial optimization. In this set of notes, we focus on the case when the underlying graph is bipartite.

We start by introducing some basic graph terminology. A *graph* $G = (V, E)$ consists of a set V of *vertices* and a set E of pairs of vertices called *edges*. For an edge $e = (u, v)$, we say that the *endpoints* of e are u and v; we also say that e is *incident* to u and v. A graph $G = (V, E)$ is *bipartite* if the vertex set V can be partitioned into two sets A and B (*the bipartition*) such that no edge in E has both endpoints in the same set of the bipartition. A matching $M \subseteq E$ is a collection of edges such that every vertex of V is incident to at most one edge of M. If a vertex v has no edge of M incident to it then v is said to be *exposed* (or unmatched). A matching is *perfect* if no vertex is exposed; in other words, a matching is perfect if its cardinality is equal to $|A| = |B|$.

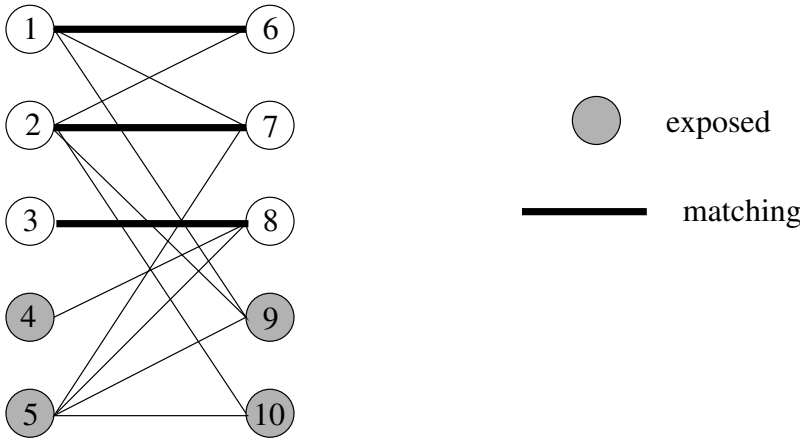


Figure 1.1: Example. The edges $(1, 6)$, $(2, 7)$ and $(3, 8)$ form a matching. Vertices 4, 5, 9 and 10 are exposed.

We are interested in the following two problems:

Maximum cardinality matching problem: Find a matching M of maximum size.

Minimum weight perfect matching problem: Given a cost c_{ij} for all $(i, j) \in E$, find a perfect matching of minimum cost where the cost of a matching M is given by $c(M) = \sum_{(i,j)\in M} c_{ij}$. This problem is also called the *assignment problem*.

Similar problems (but more complicated) can be defined on non-bipartite graphs.

1.1 Maximum cardinality matching problem

Before describing an algorithm for solving the maximum cardinality matching problem, one would like to be able to prove optimality of a matching (without reference to any algorithm). For this purpose, one would like to find upper bounds on the size of any matching and hope that the smallest of these upper bounds be equal to the size of the largest matching. This is a *duality* concept that will be ubiquitous in this subject. In this case, the dual problem will itself be a combinatorial optimization problem.

A *vertex cover* is a set C of vertices such that all edges e of E are incident to at least one vertex of C. In other words, there is no edge completely contained in $V - C$ (we use both $-$ and $\setminus$ to denote the difference of two sets). Clearly, the size of any matching is at most the size of any vertex cover. This follows from the fact that, given any matching M, a vertex cover C must contain at least one of the endpoints of each edge in M. We have just proved *weak duality*: The maximum size of a matching is at most the minimum size of a vertex cover. As we'll prove later in these notes, equality in fact holds:

Theorem 1.1 (König 1931) *For any bipartite graph, the maximum size of a matching is equal to the minimum size of a vertex cover.*

We shall prove this *minmax* relationship algorithmically, by describing an efficient algorithm which simultaneously gives a maximum matching and a minimum vertex cover. König's theorem gives a *good characterization* of the problem, namely a simple proof of optimality. In the example above, one can prove that the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$ is of maximum size since there exists a vertex cover of size 4. Just take the set $\{1, 2, 5, 8\}$.

The natural approach to solving this cardinality matching problem is to try a *greedy* algorithm: Start with any matching (e.g. an empty matching) and repeatedly add disjoint edges until no more edges can be added. This approach, however, is not guaranteed to give a maximum matching (convince yourself). We will now present an algorithm that does work, and is based on the concepts of *alternating paths* and *augmenting paths*. A path is simply a collection of edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ where the v_i's are distinct vertices. A path can simply be represented as v_0-v_1-...-v_k.

Definition 1.1 *An* **alternating path with respect to** M *is a path that alternates between edges in* M *and edges in* $E - M$*.*

Definition 1.2 *An* **augmenting path with respect to** M *is an alternating path in which the first and last vertices are exposed.*

In the above example, the paths 4-8-3, 6-1-7-2 or 5-7-2-6-1-9 are alternating, but only the last one is augmenting. Notice that an augmenting path with respect to M which contains k edges of M must also contain exactly $k + 1$ edges not in M. Also, the two endpoints of an augmenting path must be on different sides of the bipartition. The most interesting property of an augmenting path P with respect to a matching M is that if we set $M' = M \triangle P \equiv (M - P) \cup (P - M)$, then we get a matching M' and, moreover, the size of

M' is one unit larger than the size of M. That is, we can form a larger matching M' from M by taking the edges of P not in M and adding them to M' while removing from M' the edges in M that are also in the path P. We say that we have *augmented M along P*.

The usefulness of augmenting paths is given in the following theorem.

Theorem 1.2 *A matching M is maximum if and only if there are no augmenting paths with respect to M.*

Proof: (By contradiction)

($\Rightarrow$) Let P be some augmenting path with respect to M. Set $M' = M \triangle P$. Then M' is a matching with cardinality greater than M. This contradicts the maximality of M.

($\Leftarrow$) If M is not maximum, let M^* be a maximum matching (so that $|M^*| > |M|$). Let $Q = M \triangle M^*$. Then:

- Q has more edges from M^* than from M (since $|M^*| > |M|$ implies that $|M^* - M| > |M - M^*|$).
- Each vertex is incident to at most one edge in $M \cap Q$ and one edge $M^* \cap Q$.
- Thus Q is composed of cycles and paths that alternate between edges from M and M^*.
- Therefore there must be some path with more edges from M^* in it than from M (all cycles will be of even length and have the same number of edges from M^* and M). This path is an augmenting path with respect to M.

Hence there must exist an augmenting path P with respect to M, which is a contradiction. $\triangle$

This theorem motivates the following algorithm. Start with any matching M, say the empty matching. Repeatedly locate an augmenting path P with respect to M, augment M along P and replace M by the resulting matching. Stop when no more augmenting path exists. By the above theorem, we are guaranteed to have found an optimum matching. The algorithm terminates in μ augmentations, where μ is the size of the maximum matching. Clearly, $\mu \leq \frac{n}{2}$ where $n = |V|$.

In the example, one would thus augment M along an augmenting path, say 5-7-2-6-1-9, obtain the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$, and then realize that no more augmenting paths can be found.

The question now is how to decide the existence of an augmenting path and how to find one, if one exists. These tasks can be done as follows. Direct edges in G according to M as follows : An edge goes from A to B if it does not belong to the matching M and from B to A if it does. Call this directed graph D.

Claim 1.3 *There exists an augmenting path in G with respect to M iff there exists a directed path in D between an exposed vertex in A and an exposed vertex in B.*

Exercise 1-1. Prove claim 1.3.

This gives an $O(m)$ algorithm (where $m = |E|$) for finding an augmenting path in G. Let A^* and B^* be the set of exposed vertices w.r.t. M in A and B respectively. We can simply attach a vertex s to all the vertices in A^* and do a depth-first-search from s till we hit a vertex in B^* and then trace back our path.

Thus the overall complexity of finding a maximum cardinality matching is $O(nm)$. This can be improved to $O(\sqrt{n}m)$ by augmenting along several augmenting paths simultaneously.

If there is no augmenting path with respect to M, then we can also use our search procedure for an augmenting path in order to construct an optimum vertex cover. Consider the set L (for Labelling) of vertices which can be reached by a directed path from an exposed vertex in A.

Claim 1.4 *When the algorithm terminates, $C^* = (A - L) \cup (B \cap L)$ is a vertex cover. Moreover, $|C^*| = |M^*|$ where M^* is the matching returned by the algorithm.*

This claim immediately proves König's theorem.

Proof: We first show that C^* is a vertex cover. Assume not. Then there must exist an edge $e = (a, b) \in E$ with $a \in A \cap L$ and $b \in (B - L)$. The edge e cannot belong to the matching. If it did, then b should be in L for otherwise a would not be in L. Hence, e must be in $E - M$ and is therefore directed from A to B. This therefore implies that b can be reached from an exposed vertex in A (namely go to a and then take the edge (a, b)), contradicting the fact that $b \notin L$.

To show the second part of the proof, we show that $|C^*| \leq |M^*|$, since the reverse inequality is true for any matching and any vertex cover. The proof follows from the following observations.

1. No vertex in $A - L$ is exposed by definition of L,
2. No vertex in $B \cap L$ is exposed since this would imply the existence of an augmenting path and, thus, the algorithm would not have terminated,
3. There is no edge of the matching between a vertex $a \in (A-L)$ and a vertex $b \in (B \cap L)$. Otherwise, a would be in L.

These remarks imply that every vertex in C^* is matched and moreover the corresponding edges of the matching are distinct. Hence, $|C^*| \leq |M^*|$. △

Although the concepts of maximum matchings and minimum vertex covers can be defined also for general (i.e. non-bipartite) graphs, we should remark that König's theorem does not generalize to all graphs. Indeed, although it is true that the size of a maximum matching is *always* at most the minimum size of a vertex cover, equality does not necessarily hold. Consider indeed the cycle C_3 on 3 vertices (the smallest non-bipartite graph). The maximum matching has size 1, but the minimum vertex cover has size 2. We will derive a minmax relation involving maximum matchings for general graphs, but it will be more complicated than König's theorem.

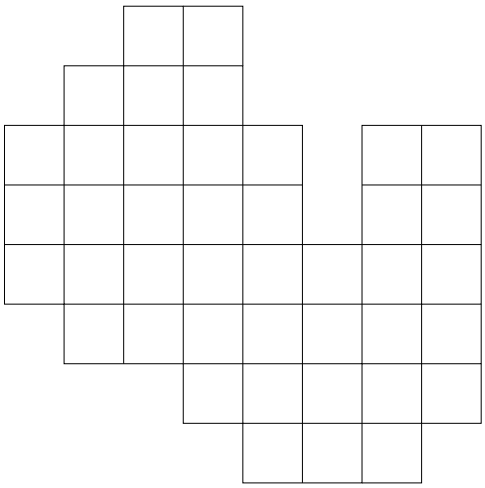
Exercises

Exercise 1-2. An *edge cover* of a graph $G = (V, E)$ is a subset of R of E such that every vertex of V is incident to at least one edge in R. Let G be a bipartite graph with no isolated vertex. Show that the cardinality of the minimum edge cover R^* of G is equal to $|V|$ minus the cardinality of the maximum matching M^* of G. Give an efficient algorithm for finding the minimum edge cover of G. Is this true also for non-bipartite graphs?

Exercise 1-3. Show that in any graph $G = (V, E)$ (not necessarily bipartite), the size of *any maximal* matching M (i.e. a matching M in which one cannot add an edge while keeping it a matching) is at least half the size of a *maximum* matching M^*.

Exercise 1-4. Consider the problem of perfectly tiling a subset of a checkerboard (i.e. a collection of unit squares, see example below) with dominoes (a domino being 2 adjacent squares).

1. Show that this problem can be formulated as the problem of deciding whether a bipartite graph has a perfect matching.
2. Can the following figure be tiled by dominoes? Give a tiling or a short proof that no tiling exists.



Exercise 1-5. Consider a bipartite graph $G = (V, E)$ with bipartition (A, B): $V = A \cup B$. Assume that, for some vertex sets $A_1 \subseteq A$ and $B_1 \subseteq B$, there exists a matching M_A covering all vertices in A_1 and a matching M_B covering all vertices in B_1. Prove that there always exists a matching covering all vertices in $A_1 \cup B_1$.

Exercise 1-6. Consider the following 2-person game on a (not necessarily bipartite) graph $G = (V, E)$. Players 1 and 2 alternate and each selects a (yet unchosen) edge e of the graph so that e together with the previously selected edges form a simple path. The first player unable to select such an edge loses. Show that if G has a *perfect* matching then player 1 has a winning strategy.

1.1.1 Hall's Theorem

Hall's theorem gives a necessary and sufficient condition for a bipartite graph to have a matching which saturates (or matches) all vertices of A (i.e. a matching of size $|A|$).

Theorem 1.5 (Hall 1935) *Given a bipartite graph $G = (V, E)$ with bipartition A, B ($V = A \cup B$), G has a matching of size $|A|$ if and only if for every $S \subseteq A$ we have $|N(S)| \geq |S|$, where $N(S) = \{b \in B : \exists a \in S \text{ with } (a, b) \in E\}$.*

Clearly, the condition given in Hall's theorem is necessary; its sufficiency can be derived from König's theorem.

Exercise 1-7. Deduce Hall's theorem from König's theorem.

Exercise 1-8. Consider a bipartite graph $G = (V, E)$ with bipartition (A, B). For $X \subseteq A$, define $\text{def}(X) = |X| - |N(X)|$ where $N(X) = \{b \in B : \exists a \in X \text{ with } (a, b) \in E\}$. Let

$$\text{def}_{max} = \max_{X \subseteq A} \text{def}(X).$$

Since $\text{def}(\emptyset) = 0$, we have $\text{def}_{max} \geq 0$.

1. Generalize Hall's theorem by showing that the maximum size of a matching in a bipartite graph G equals $|A| - \text{def}_{max}$.
2. For any 2 subsets $X, Y \subseteq A$, show that

$$\text{def}(X \cup Y) + \text{def}(X \cap Y) \geq \text{def}(X) + \text{def}(Y).$$

1.2 Minimum weight perfect matching

By assigning infinite costs to the edges not present, one can assume that the bipartite graph is complete. The minimum cost (weight) *perfect* matching problem is often described by the following story: There are n jobs to be processed on n machines or computers and one would like to process exactly one job per machine such that the total cost of processing the jobs is minimized. Formally, we are given costs $c_{ij} \in \mathbb{R} \cup \{\infty\}$ for every $i \in A, j \in B$ and the goal is to find a perfect matching M minimizing $\sum_{(i,j) \in M} c_{ij}$.

In these notes, we present an algorithm for this problem which is based upon linear programming, and we will take this opportunity to illustrate several important concepts in linear programming. These concepts will be formalized and generalized in a subsequent chapter.

The first algorithm given for the assignment problem was given by Kuhn [1955], but he showed only finiteness of the algorithm. A refined version was given by Jim Munkres [1957], and showed a polynomial running time. An algorithm is polynomial-time if its running time (the number of basic operations to run it) is upper bounded by a polynomial in the *size of*

the input (i.e. the number of bits needed to represent the input). Munkres' analysis even shows that the algorithm is *strongly polynomial*, and this means that the running time is polynomial in the *number of numbers* involved (i.e. does not depend on the size of the costs c_{ij}). In this algorithm, the number of operations is upper bounded by $O(n^3)$ where $n = |V|$.

The algorithm is often called the Hungarian method, as it relies on ideas developed by Hungarians, and especially König and Egerváry. In 2006, it was discovered that the method had actually been discovered in the 19th century by Jacobi and this was posthumously published in 1890 in Latin, see http://www.lix.polytechnique.fr/∼ollivier/JACOBI/jacobiEngl.htm.

We start by giving a formulation of the problem as an *integer program*, i.e. an optimization problem in which the variables are restricted to integer values and the constraints and the objective function are linear as a function of these variables. We first need to associate a point to every matching. For this purpose, given a matching M, let its *incidence vector* be x where $x_{ij} = 1$ if $(i, j) \in M$ and 0 otherwise. One can formulate the minimum weight perfect matching problem as follows:

$$\begin{aligned} \text{Min} \quad & \sum_{i,j} c_{ij} x_{ij} & \\ \text{subject to:} & & \\ & \sum_j x_{ij} = 1 & i \in A \\ & \sum_i x_{ij} = 1 & j \in B \\ & x_{ij} \geq 0 & i \in A, j \in B \\ & x_{ij} \text{ integer} & i \in A, j \in B. \end{aligned}$$

This is not a linear program, but a so-called integer program. Notice that any solution to this integer program corresponds to a matching and therefore this is a valid formulation of the minimum weight perfect matching problem in bipartite graphs.

Consider now the linear program (P) obtained by dropping the integrality constraints:

$$(P) \qquad \begin{aligned} \text{Min} \quad & \sum_{i,j} c_{ij} x_{ij} & \\ \text{subject to:} & & \\ & \sum_j x_{ij} = 1 & i \in A \\ & \sum_i x_{ij} = 1 & j \in B \\ & x_{ij} \geq 0 & i \in A, j \in B. \end{aligned}$$

This is the linear programming *relaxation* of the above integer program. In a linear program, the variables can take fractional values and therefore there are many feasible solutions to the set of constraints above which do not correspond to matchings. But we only care

about the *optimum* solutions. The set of feasible solutions to the constraints in (P) forms a *bounded polyhedron* or *polytope*, and when we optimize a linear constraint over a polytope, the optimum will be attained at one of the "corners" or *extreme points* of the polytope. An extreme point x of a set Q is an element $x \in Q$ which cannot be written as $\lambda y + (1-\lambda)z$ with $0 < \lambda < 1$, $y, z \in Q$ with $y \neq z$. (This will be formalized and discussed in greater depth in the chapter on polyhedral combinatorics.)

In general, even if all the coefficients of the constraint matrix in a linear program are either 0 or 1, the extreme points of a linear program are not guaranteed to have all coordinates integral (this is of no surprise since the general integer programming problem is NP-hard, while linear programming is polynomially solvable). As a result, in general, there is no guarantee that the value Z_{IP} of an integer program is equal to the value Z_{LP} of its LP relaxation. However, since the integer program is *more* constrained than the relaxation, we always have that $Z_{IP} \geq Z_{LP}$, implying that Z_{LP} is a lower bound on Z_{IP} for a minimization problem. Moreover, if an optimum solution to a linear programming relaxation is integral (in our case, that would imply it is the incidence vector of a perfect matching) then it must also be an optimum solution to the integer program.

Exercise 1-9. Prove this last claim.

Exercise 1-10. Give an example of an integer program where $Z_{IP} \neq Z_{LP}$.

However, in the case of the perfect matching problem, the constraint matrix has a very special form and one can show the following very important result.

Theorem 1.6 *Any extreme point of (P) is a 0-1 vector and, hence, is the incidence vector of a perfect matching.*

Because of the above theorem, the polytope

$$\begin{aligned} P \quad = \quad \{x : \ & \sum_j x_{ij} = 1 && i \in A \\ & \sum_i x_{ij} = 1 && j \in B \\ & x_{ij} \geq 0 && i \in A, j \in B\} \end{aligned}$$

is called the *bipartite perfect matching polytope*.

To demonstrate the beauty of matchings, we shall give two completely different proofs of this result, one purely algorithmic here and one purely algebraic in the chapter on polyhedral theory. The algebraic proof is related to the notion of *totally unimodularity*.

To prove it algorithmically, we describe an algorithm for solving the minimum weight perfect matching problem. The algorithm is "primal-dual". To explain what this means, we need to introduce the notion of duality of linear programs, and let's do it in the specific case of our bipartite matching problem. Suppose we have values u_i for $i \in A$ and v_j for $j \in B$

such that $u_i + v_j \leq c_{ij}$ for all $i \in A$ and $j \in B$. Then for any perfect matching M, we have that

$$\sum_{(i,j)\in M} c_{ij} \geq \sum_{i\in A} u_i + \sum_{j\in B} v_j. \tag{1}$$

Thus, $\sum_{i\in A} u_i + \sum_{j\in B} v_j$ is a *lower bound* on the cost of the minimum cost perfect matching (for bipartite graphs). To get the best lower bound, we would like to maximize this quantity, and therefore we obtain another linear program

$$\begin{aligned} &\text{Max} && \sum_{i\in A} u_i + \sum_{j\in B} v_j && \\ &\text{subject to:} && && \\ (D) \qquad & && u_i + v_j \leq c_{ij} && i \in A, j \in B. \end{aligned}$$

The constraints can be interpreted as $w_{ij} \geq 0$ where $w_{ij} = c_{ij} - u_i - v_j$. This is a linear program, call it (D). So far, we have shown that this linear program (D) provides a lower bound on the cost of any perfect matching, but we can even prove that it provides a lower bound on the value of any solution to the linear program (P). Indeed consider any $x \in P$. We have

$$\begin{aligned} \sum_{i\in A}\sum_{j\in B} c_{ij}x_{ij} &\geq \sum_{i\in A}\sum_{j\in B}(u_i + v_j)x_{ij} \\ &= \left(\sum_{i\in A} u_i \sum_{j\in B} x_{ij}\right) + \left(\sum_{j\in B} v_j \sum_{i\in A} xij\right) \\ &= \sum_{i\in A} u_i + \sum_{j\in B} v_j, \end{aligned}$$

because of the constraints that x satisfy. (D) is the *dual* linear program in the sense of linear programming duality.

In summary, so far, we know that

$$\left[\min_{\text{perfect matchings } M} \sum_{(i,j)\in M} c_{ij}\right] \geq \left[\min_{x\in P} \sum_{(i,j)} c_{ij}x_{ij}\right] \geq \left[\max_{(u,v)\in D} \sum_{i\in A} u_i + \sum_{j\in B} v_j\right].$$

If, for any instance, we could always find a feasible solution u, v to (D) and a perfect matching M such that we have equality in (1) (i.e. the cost of the perfect matching is equal to the value of the dual solution) then we would know that we have equality throughout, that the matching found is *optimum*, and that furthermore, the incidence vector of the matching M is optimum for the linear program (P). Given a solution u, v to the dual, a perfect matching M would satisfy equality if it contains only edges (i, j) such that $w_{ij} = c_{ij} - u_i - v_j = 0$. This is what is referred to as *complementary slackness*. However, for a given u, v, we may not be able to find a *perfect* matching among the edges with $w_{ij} = 0$.

The algorithm performs a series of iterations to obtain an appropriate u and v. It always maintains a dual feasible solution and tries to find an "almost" primal feasible solution x satisfying complementary slackness. The fact that complementary slackness is imposed is crucial in any primal-dual algorithm. In fact, the most important (and elegant) algorithms in combinatorial optimization are primal-dual. This is one of the most important tool for designing efficient algorithms for combinatorial optimization problems (for problems which, of course, admit such efficient solutions).

More precisely, the algorithm works as follows. It first starts with any dual feasible solution, say $u_i = 0$ for all i and $v_j = \min_i c_{ij}$ for all j. In a given iteration, the algorithm has a dual feasible solution (u, v) or say (u, v, w). Imposing complementary slackness means that we are interested in matchings which are subgraphs of $E = \{(i, j) : w_{ij} = 0\}$. If E has a perfect matching then the incidence vector of that matching is a feasible solution in (P) and satisfies complementary slackness with the current dual solution and, hence, must be optimal. To check whether E has a perfect matching, one can use the cardinality matching algorithm developed earlier in these notes. If the maximum matching output is not perfect then the algorithm will use information from the optimum vertex cover C^* to update the dual solution in such a way that the value of the dual solution increases (we are maximizing in the dual).

In particular, if L is as in the previous section then there is no edge of E between $A \cap L$ and $B - L$. In other words, for every $i \in (A \cap L)$ and every $j \in (B - L)$, we have $w_{ij} > 0$. Let

$$\delta = \min_{i \in (A \cap L), j \in (B-L)} w_{ij}.$$

By the above argument, $\delta > 0$. The dual solution is updated as follows:

$$u_i = \begin{cases} u_i & i \in A - L \\ u_i + \delta & i \in A \cap L \end{cases}$$

and

$$v_j = \begin{cases} v_j & i \in B - L \\ v_j - \delta & j \in B \cap L \end{cases}$$

One easily check that this dual solution is feasible, in the sense that the corresponding vector w satisfies $w_{ij} \geq 0$ for all i and j. What is the value of the new dual solution? The difference between the values of the new dual solution and the old dual solution is equal to:

$$\delta(|A \cap L| - |B \cap L|) = \delta(|A \cap L| + |A - L| - |A - L| - |B \cap L|) = \delta(\frac{n}{2} - |C^*|),$$

where A has size $n/2$ and C^* is the optimum vertex cover for the bipartite graph with edge set E. But by assumption $|C^*| < \frac{n}{2}$, implying that the value of the dual solution strictly increases.

One repeats this procedure until the algorithm terminates. At that point, we have an incidence vector of a perfect matching and also a dual feasible solution which satisfy complementary slackness. They must therefore be optimal and this proves the existence of

an *integral* optimum solution to (P). Since, by carefully choosing the cost function, one can make any extreme point be the *unique* optimum solution to the linear program (this will be formally proved in the polyhedral chapter), this shows that any extreme point is integral and hence this proves Theorem 1.6.

Of course, as some of the readers might have noticed, the proof is not complete yet since one needs to prove that the algorithm indeed terminates. This can be proved by noticing that at least one more vertex of B must be reachable from an exposed vertex of A (and no vertex of B becomes unreachable), since an edge $e = (i, j)$ with $i \in (A \cap L)$ and $j \in B - L$ now has $w_{ij} = 0$ by our choice of δ. This also gives an estimate of the number of iterations. In at most $n/2$ iterations, all vertices of B are reachable or the matching found has increased by at least one unit. Therefore, after $O(n^2)$ iterations, the matching found is perfect. The overall running time of the algorithm is thus $O(n^4)$ since it takes $O(n^2)$ to compute the set L in each iteration. By looking more closely at how vertices get labelled between two increases of the size of the matching, one can reduce the running time analysis to $O(n^3)$.

Exercise 1-11. Check that the running time of the algorithm is indeed $O(n^3)$.

Example: Consider the instance given by the following cost matrix defined on a bipartite graph with 5 vertices on each side of the bipartition:

0	2	7	2	3
1	3	9	3	3
1	3	3	1	2
4	0	1	0	2
0	0	3	0	0

Assume that $u^T = (2, 3, 0, -2, 0)$ and $v^T = (-2, 0, 3, 0, 0)$. The set E of edges with $w_{ij} = 0$ corresponds exactly to the set of edges in Figure 1.1. The maximum cardinality matching algorithm finds the matching $(1, 9)$, $(2, 6)$, $(3, 8)$ and $(5, 7)$, and the set of labelled vertices is $\{3, 4, 8\}$. We compute δ as

$$\delta = \min_{i \in \{3,4\}, j \in \{6,7,9,10\}} w_{ij} = 1$$

corresponding to the edge $(3, 9)$. The new vectors u and v are $u^T = (2, 3, 1, -1, 0)$ and $v^T = (-2, 0, 2, 0, 0)$. The value of the dual solution has increased from 4 units to 5. The corresponding set E now has a perfect matching, namely $(1, 6)$, $(2, 7)$, $(3, 9)$, $(4, 8)$ and $(5, 10)$ of cost 5. Both the matching and the dual solution are optimal.

Exercises

Exercise 1-12. Consider a bipartite graph $G = (V, E)$ in which every vertex has degree k (a so-called k-regular bipartite graph). Prove that such a graph always has a perfect matching in two different ways:

1.by using König's theorem,

2.by using the linear programming formulation we have derived in this section.

(Optional: Is this result also true for non-bipartite graphs?)

Exercise 1-13. Using the previous exercise 12, show that the edges of a k-regular bipartite graph G can be partitioned into k matchings (i.e. the number of colors needed to color the edges of a k-regular bipartite graph such that no two edges with a common endpoint have the same color — *the edge chromatic number* — is precisely k).

(Optional: Is this result also true also for non-bipartite graphs?)

Exercise 1-14. Given a graph $G = (V, E)$, its edge coloring number is the smallest number of colors needed to color the edges in E so that any two edges having a common endpoint have a different color.

1.Show that the edge coloring number of a *bipartite* graph G is always equal to its maximum degree Δ (i.e. the maximum over all vertices v of the number of edges incident to v). (Use the previous problem.)

2.Give an example of a non-bipartite graph for which the edge coloring number is (strictly) greater than Δ.

Exercise 1-15. We have shown that there always exists a solution x to the linear program (P) with all components integral. Reprove this result in the following way.

Take a (possibly non-integral) *optimum* solution x^*. If there are many optimum solutions, take one with as few non-integral values x^*_{ij} as possible. Show that, if x^* is not integral, there exists a cycle C with all edges $e = (i, j) \in C$ having a non-integral value x^*_{ij}. Now show how to derive another optimum solution with fewer non-integral values, leading to a contradiction.

Exercise 1-16. In this exercise, you will do a little experiment with the (minimum cost) assignment problem. Take a complete bipartite graph with n vertices on each side of the bipartition, and let us assume that all c_{ij} (for $i, j \in \{1, \cdots, n\}$) are all independent uniform random variables between 0 and 1. Take 5 different values for n (the largest being a few hundreds) and for each compute the *minimum* cost assignment value for 5 instances. Any guess on how this value increases as n goes to infinity. Does it seem to converge? To what value? Surprised? (Yes, you should be, and it is normal that you do not understand why.)

To solve the assignment problem, you can use MATLAB (it is available on athena). To access MATLAB on athena, you need to type:

```
>> add matlab
>> matlab
```

If you have never used MATLAB before, you can find some tutorial on the 18.06 webpage. There is a MATLAB m-file at http://www-math.mit.edu/∼goemans/bghungar.m which implements the Hungarian method (just put it in the directory from which you run MATLAB). If C is an $n \times n$ matrix, then

```
>> a=bghungar(C)
```

gives a vector a such that $(1, a(1)), (2, a(2)), \cdots$ is the *maximizing* matching. So to get the value of the *minimum* assignment, you can just do:

```
>> a=bghungar(-C);
>> v=sum(diag(C(1:n,a(1:n))))
```

Exercise 1-17. For the assignment problem, the greedy algorithm (which repeatedly finds the minimum cost edge disjoint from all the previously selected edges) can lead to a solution whose cost divided by the optimum cost can be arbitrarily large (even for graphs with 2 vertices on each side of the bipartition).

Suppose now that the cost comes from a metric, even just a line metric. More precisely, suppose that the bipartition is $A \cup B$ with $|A| = |B| = n$ and the ith vertex of A (resp. the jth vertex of B) is associated with $a_i \in \mathbb{R}$ (resp. $b_j \in B$). Suppose that the cost between these vertices is given by $c_{ij} = |a_i - b_j|$.

Consider the greedy algorithm: select the closest pair of vertices, one from A and from B, match them together, delete them, and repeat until all vertices are matched. For these line metric instances, is the cost of the greedy solution always upper bounded by a constant (independent of n) times the optimum cost of the assignment? If so, prove it; if not, give a family of examples (parametrized by n) such that the corresponding ratio becomes arbitrarily large.

2. Lecture notes on non-bipartite matching

Given a graph $G = (V, E)$, we are interested in finding and charaterizing the size of a maximum matching. Since we do not assume that the graph is bipartite, we know that the maximum size of a matching does not necessarily equal the minimum size of a vertex cover, as it is the case for bipartite graphs (König's theorem). Indeed, for a triangle, any matching consists of at most one edge, while we need two vertices to cover all edges.

To get an upper bound on the size of any matching M, consider any set U of vertices. If we delete the vertices in U (and all edges adjacent to it), we delete at most $|U|$ edges of the matching M. Moreover, in the remaining graph $G \setminus U$, we can trivially upper bound the size of the remaining matching by $\sum_{i=1}^{k} \lfloor \frac{|K_i|}{2} \rfloor$, where K_i, $i = 1, \cdots, k$, are the vertex sets of the connected components of $G \setminus U$. Therefore, we get that

$$|M| \leq |U| + \sum_{i=1}^{k} \left\lfloor \frac{|K_i|}{2} \right\rfloor . \tag{1}$$

If we let $o(G \setminus U)$ denote the number of odd components of $G \setminus U$, we can rewrite (1) as:

$$|M| \leq |U| + \frac{|V| - |U|}{2} - \frac{o(G \setminus U)}{2},$$

or

$$|M| \leq \frac{1}{2} \left(|V| + |U| - o(G \setminus U) \right). \tag{2}$$

We will show that we can always find a matching M and a set U for which we have equality; this gives us the following minmax relation, called the Tutte-Berge min-max formula:

Theorem 2.1 (Tutte-Berge Formula) *For any graph $G = (V, E)$, we have*

$$\max_{M} |M| = \min_{U \subseteq V} \frac{1}{2} \left(|V| + |U| - o(G \setminus U) \right),$$

where $o(G \setminus U)$ is the number of connected components of odd size of $G \setminus U$.

Example: In the graph of Figure 2.1, a matching of size 8 can be easily found (find it), and its optimality can be seen from the Tutte-Berge formula. Indeed, for the set $U = \{2, 15, 16\}$, we have $o(G \setminus U) = 5$ and $\frac{1}{2}(|V| + |U| - o(G \setminus U)) = \frac{1}{2}(18 + 3 - 5) = 8$.

To prove Theorem 2.1, we will first show an algorithm to find a maximum matching. This algorithm is due to Edmonds [1965], and is a pure gem. As in the case of bipartite matchings (see lecture notes on bipartite matchings), we will be using augmenting paths. Indeed, Theorem 1.2 of the bipartite matching notes still hold in the non-bipartite setting; a matching M is maximum if and only if there is no augmenting path with respect to it. The difficulty here is to find the augmenting path or decide that no such path exists. We could try to start from the set X of exposed (unmatched) vertices for M, and whenever we are

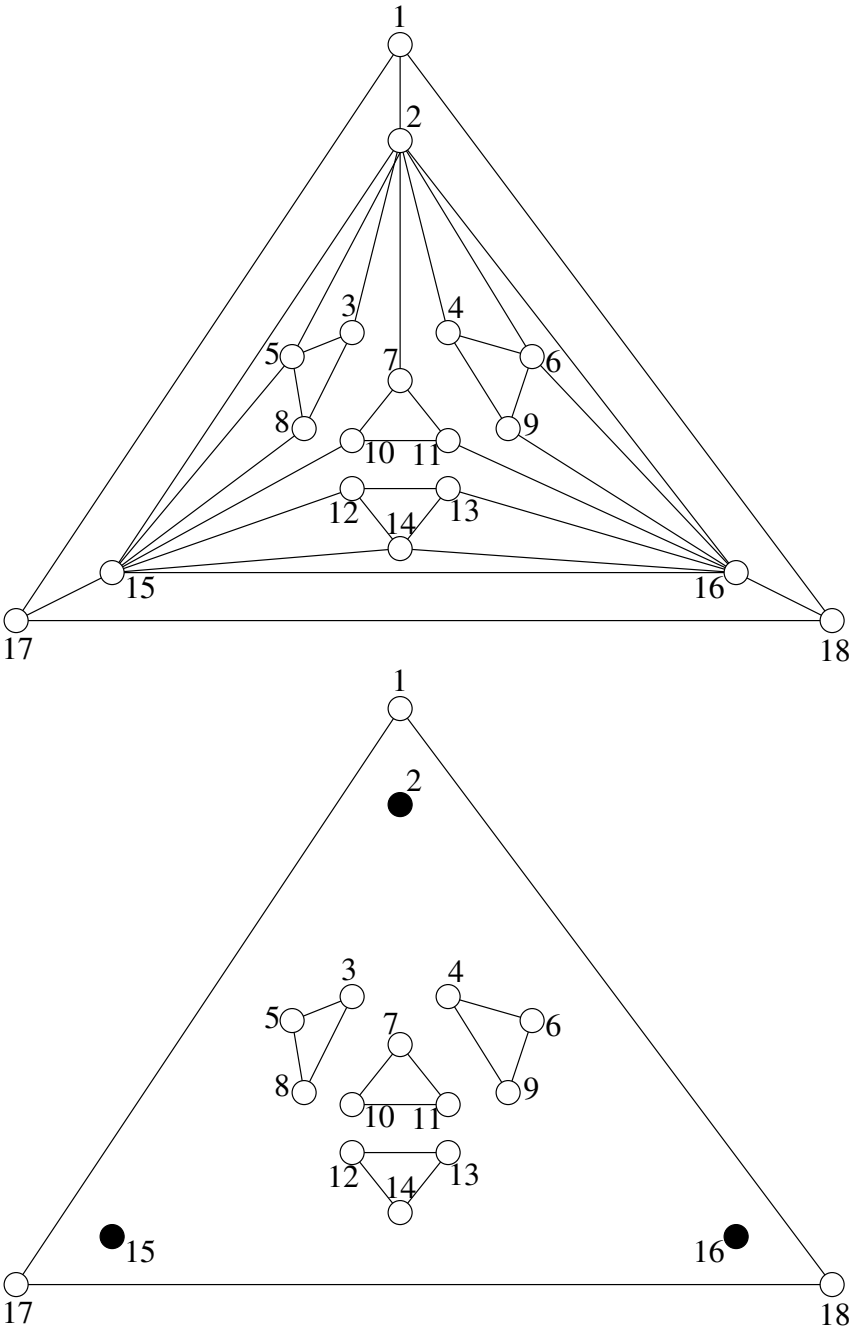


Figure 2.1: Top: graph. Bottom: the removal of vertices 2, 15 and 16 gives 5 odd connected components.

Figure 2.2: An augmenting path. Thick edges are in the matching. The path is found by starting from the exposed vertex u and following the dotted lines until a vertex (here z) adjacent to an exposed vertex is found.

at a vertex u and see an edge $(u, v) \notin M$ followed by an edge (v, w) in M, we could put a directed edge from u to w and move to w. If we get to a vertex that's adjacent to an exposed vertex (i.e. in X), it seems we have found an augmenting path, see Figure 2.2.

This is not necessarily the case, as the vertices of this 'path' may not be distinct. We have found a so-called *flower*, see Figure 2.3. This flower does not contain an augmenting path. More formally, a *flower* consists of an even alternating path P from an exposed vertex u to a vertex v, called the *stem*, and an odd cycle containing v in which the edges alternate between in and out of the matching except for the two edges incident to v; this odd cycle is called a *blossom*.

The algorithm will either find an augmenting path or a flower or show that no such items exist; in this latter case, the matching is maximum and the algorithm stops. If it finds an augmenting path then the matching is augmented and the algorithm continues with this new matching. If a flower is found, we create a new graph G/B in which we shrink B into a single vertex b; any edge (u, v) in G with $u \notin B$ and $v \in B$ is replaced by an edge (u, b) in G/B, all edges within B disappear and all edges within $V \setminus B$ are kept. Notice that we have also a matching M/B in this new graph (obtained by simply deleting all edges of M within B), and that the sizes of M and M/B differ by exactly $\frac{|B|-1}{2}$ (as we deleted so many edges of the matching within B). We use the following crucial theorem.

Theorem 2.2 *Let B be a blossom with respect to M. Then M is a maximum size matching in G if and only if M/B is a maximum size matching in G/B.*

To prove the theorem, we can assume that the flower with blossom B has an empty stem P. If it is not the case, we can consider the matching $M \triangle P = (M \setminus P) \cup (P \setminus M)$ for which we have a flower with blossom P and empty stem. Proving the theorem for $M \triangle P$ also proves it for M as $(M \triangle P)/B = (M/B) \triangle P$ and taking symmetric differences with an even alternating paths does not change the cardinality of a matching.

Proof: ($\Longrightarrow$) Suppose N is a matching in G/B larger than M/B. Pulling N back to a set of edges in G, it is incident to at most one vertex of B. Expand this to a matching N^+ in G by adjoining $\frac{1}{2}(|B| - 1)$ edges to match every other vertex in B. Then $|N^+|$ exceeds $|M|$ by the same amount that $|N|$ exceeds $|M/B|$.

($\Longleftarrow$) By contradiction. If M is not of maximum size in G then it has an augmenting path P between exposed vertices u and v. As B has only one exposed vertex, we can assume that $u \notin B$. Let w be the first vertex of P which belongs to B, and let Q be the part of P

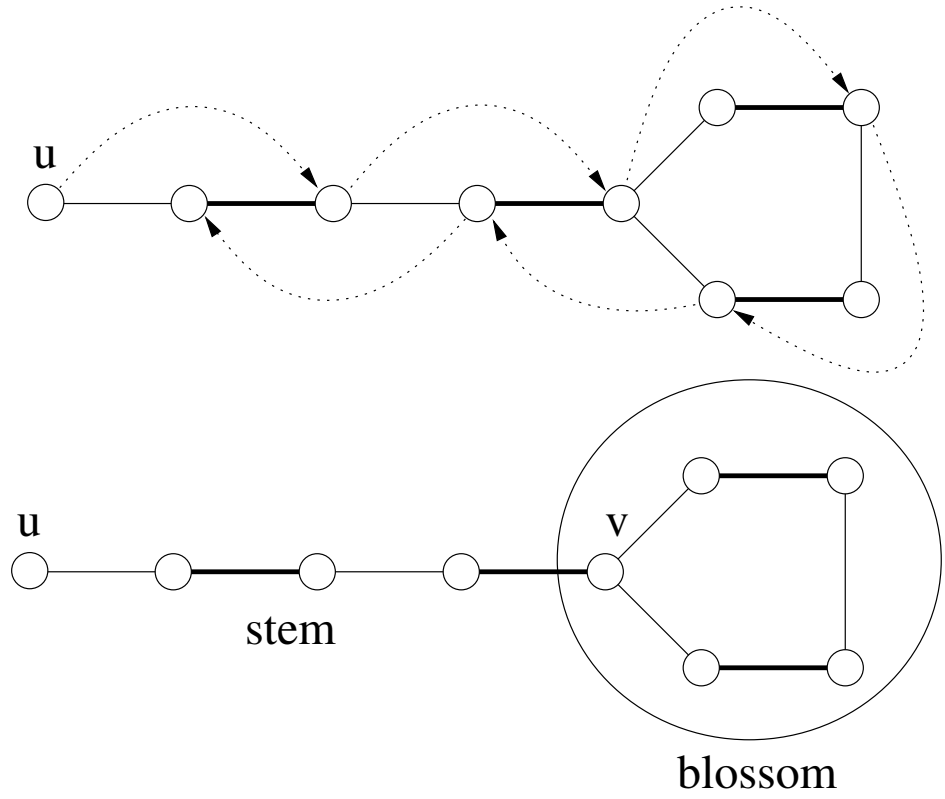


Figure 2.3: A flower. The thick edges are those of the matching. Top: Our dotted path starting at an exposed vertex u and ending at a neighbor of an exposed vertex does not correspond to an augmenting path.

from u to w. Notice that, after shrinking B, Q remains an augmenting path for M/B (since b is exposed in G/B). This means that M/B is not maximum either, and we have reached a contradiction. △

Observe that the proof is algorithmic: If we have a bigger matching in G/B than M/B then we also can find a bigger matching in G than M. Also remark that Theorem 2.2 does *not* say that if we find a maximum matching M^* in G/B then simply adding $\frac{|B|-1}{2}$ edges from within B to M^* to get $\hat{M}$ will lead to a *maximum* matching in G. Indeed, this is not true.

Exercise 2-1. Give an example of a graph G, a matching M and a blossom B for M such that a maximum matching M^* in G/B does not lead to a maximum matching in G. Explain why this does not contradict Theorem 2.2.

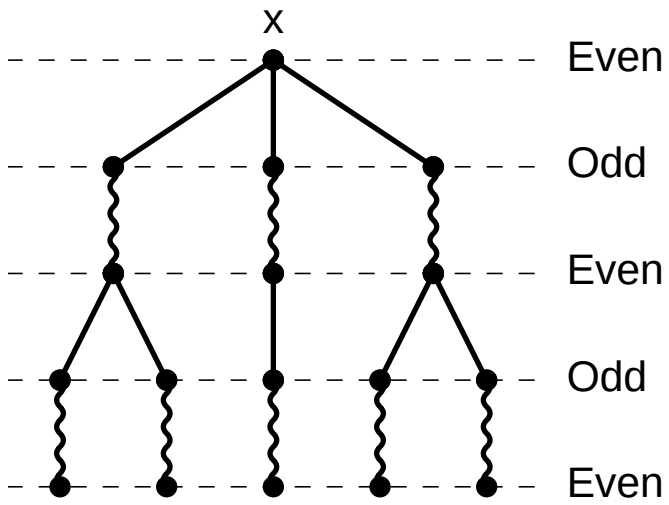


Figure 2.4: An alternating tree. The squiggly edges are the matching edges.

To find either an alternating path or a flower, we proceed as follows. We label all exposed vertices to be EVEN, and keep all the other vertices unlabelled at this point. As we proceed, we will be labelling more vertices to be EVEN as well as labelling some vertices to be ODD. We maintain also an *alternating forest* — a graph in which each connected component is a tree made up of edges alternating between being in and out of the matching. We process the EVEN vertices one at a time, say we are currently processing u, and consider the edges adjacent to u. There are several possibilities:

1. If there is an edge (u, v) with v unlabelled, we label v as ODD. As v cannot be exposed (as otherwise it would have been already EVEN), we label its "mate" w (i.e. (v, w) is an edge of the matching) as EVEN. (w was not previously labelled as we always simultaneously label the two endpoints of a matched edge.) We have extended the alternating tree we are building (see Figure 2.4).

2. If there is an edge (u, v) with v labelled EVEN and v belongs to another alternating tree than u does, we have found an augmenting path (just traverse the 2 alternating trees from u and v up to their roots) and augment the matching along it, and start again from this new, larger matching. The two subpaths from u and from v to their roots span disjoint sets of vertices, and therefore their union together with (u, v) indeed form a valid augmenting path.

3. If there is an edge (u, v) with v labelled EVEN and v belongs to the same alternating tree as u does, then the two subpaths from u and v to their common (exposed) root x together with (u, v) form a flower. We shrink the blossom B into a vertex b. Observe that we can keep our labelling unchanged, provided we let the new vertex b be labelled EVEN. We recursively find a maximum matching in this graph G/B (and this may result in further shrinkings) and when the algorithm terminates, we use Theorem 2.2 to expand it to a larger matching in the original graph. This larger matching is not necessarily optimal (see the remark after Theorem 2.2) and we repeat the process to find either an augmenting path or a flower with respect to the current matching.

Correctness. Now suppose that none of these possibilities apply any more for any of the EVEN vertices. Then we claim that we have found a maximum matching M' in the current graph $G' = (V', E')$ (which was obtained from our original graph G by performing several shrinkings of blossoms $B_1, B_2, \cdots, B_k$ in succession). To show this, consider $U = \text{ODD}$ and consider the upper bound (2) for G'. As there are no edges between EVEN vertices (otherwise 2. or 3. above would apply) and no edges between an EVEN vertex and an unlabelled vertex (otherwise 1. would apply), we have that each EVEN vertex is an (odd-sized) connected component by itself in $G' \setminus \text{ODD}$. Thus $o(G' \setminus \text{ODD}) = |\text{EVEN}|$. Also, we have that $|M'| = |\text{ODD}| + \frac{1}{2}(|V'| - |\text{ODD}| - |\text{EVEN}|)$, the second term coming from the fact that all unlabelled vertices are matched. Thus,

$$\frac{1}{2}(|V'| + |\text{ODD}| - o(G' \setminus \text{ODD})) = \frac{1}{2}(|V'| + |\text{ODD}| - |\text{EVEN}|) = |M'|,$$

and this shows that our matching M' is maximum for G'. Applying repeatedly Theorem 2.2, we get that the algorithm constructs a maximum matching in G.

Running Time. The algorithm will perform at most n augmentations (of the matching) where $n = |V|$. Between two augmentations, it will shrink a blossom at most $n/2$ times, as each shrinking reduces the number of vertices by at least 2. The time it takes to construct the alternating tree is at most $O(m)$ where $m = |E|$, and so the total time is $O(n^2 m)$.

Correctness of Tutte-Berge Formula. We can now prove Theorem 2.1. As we have argued the Tutte-Berge formula holds for the graph obtained at the end of the algorithm. Assume we have performed k blossom shrinkings, and let $G_i = (V_i, E_i)$ be the graph obtained after shrinking blossoms $B_1, \cdots, B_i$, and let M_i be the corresponding matching; the index

$i = 0$ corresponds to the original graph. For the final graph $G_k = (V_k, E_k)$, we have seen that the Tutte-Berge formula holds since

$$|M_k| = \frac{1}{2}\left(|V_k| + |U| - o(G_k \setminus U)\right),$$

where $U = \textsc{Odd}$, and that each Even vertex corresponds to an odd connected component of $G_k \setminus U$. Now, let's see what happens when we unshrink blossoms, one at a time, and let's proceed by backward induction. Suppose we unshrink blossom B_i to go from graph G_i to G_{i-1}. First notice that $|V_{i-1}| = |V_i| + |B_i| - 1$ and $|M_{i-1}| = |M_i| + \frac{1}{2}(|B_i| - 1)$. Also, as we unshrink blossom B_i, we add an even number of vertices (namely $|B_i| - 1$) to one of the connected components of $G_i \setminus U$, and therefore we do not change the number of odd (or even) connected components. Thus, $o(G_i \setminus U) = o(G_{i-1} \setminus U)$. Thus, as we replace i with $i - 1$, both the right-hand-side and left-hand-side of

$$|M_i| = \frac{1}{2}\left(|V_i| + |U| - o(G_i \setminus U)\right)$$

increase by precisely $\frac{1}{2}(|B_i| - 1)$. Thus, by backward induction, we can show that for every $j = 0, \cdots, k$, we have

$$|M_j| = \frac{1}{2}\left(|V_j| + |U| - o(G_j \setminus U)\right),$$

and the Tutte-Berge formula holds for the original graph (for $j = 0$). This proves Theorem 2.1.

The Tutte-Berge formula implies that a graph has a perfect matching if and only if for every set U the number of odd connected components of $G \setminus U$ is at most $|U|$. This is known as **Tutte's matching theorem.**

Theorem 2.3 (Tutte's matching theorem) *G has a perfect matching if and only if, for all $U \subseteq V$, we have $o(G \setminus U) \leq |U|$.*

Exercises

Exercise 2-2. Let $G = (V, E)$ be any graph. Given a set $S \subseteq V$, suppose that there exists a matching M covering S (i.e. S is a subset of the matched vertices in M). Prove that there exists a *maximum* matching M^* covering S as well.

Exercise 2-3. Let U be any minimizer in the Tutte-Berge formula. Let $K_1, \cdots, K_k$ be the connected components of $G \setminus U$. Show that, for *any* maximum matching M, we must have that

1. M contains exactly $\lfloor \frac{|K_i|}{2} \rfloor$ edges from $G[K_i]$ (the subgraph of G induced by the vertices in K_i), i.e. $G[K_i]$ is perfectly matched for the even components K_i and near-perfectly matched for the odd components.
2. Each vertex $u \in U$ is matched to a vertex v in an odd component K_i of $G \setminus U$.

3.the only unmatched vertices must be in odd components K_i of $G \setminus U$.

Exercise 2-4. Could there be several minimizers U in the Tutte-Berge formula? Either give an example with several sets U achieving the minimum, or prove that the set U is unique.

Exercise 2-5. Given a graph $G = (V, E)$, an *inessential* vertex is a vertex v such that there exists a *maximum* matching of G not covering v. Let B be the set of all inessential vertices in G (e.g., if G has a perfect matching then $B = \emptyset$). Let C denote the set of vertices not in B but adjacent to at least one vertex in B (thus, if $B = \emptyset$ then $C = \emptyset$). Let $D = V \setminus (B \cup C)$. The triple $\{B, C, D\}$ is called the Edmonds-Gallai partition of G. Show that $U = C$ is a minimizer in the Tutte-Berge formula. (In particular, this means that in the Tutte-Berge formula we can assume that U is such that the union of the odd connected components of $G \setminus U$ is precisely the set of inessential vertices.)

Exercise 2-6. Show that any 3-regular 2-edge-connected graph $G = (V, E)$ (not necessarily bipartite) has a perfect matching. (A 2-edge-connected graph has at least 2 edges in every cutset; a cutset being the edges between S and $V \setminus S$ for some vertex set S.)

Exercise 2-7. A graph $G = (V, E)$ is said to be *factor-critical* if, for all $v \in V$, we have that $G \setminus \{v\}$ contains a perfect matching. In parts (a) and (b) below, G is a factor-critical graph.

1.Let U be *any* minimizer in the Tutte-Berge formula for G. Prove that $U = \emptyset$. (Hint: see Exercise 2-3.)

2.Deduce that when Edmonds algorithm terminates the final graph (obtained from G by shrinking flowers) must be a single vertex.

3.Given a graph $H = (V, E)$, an *ear* is a path $v_0 - v_1 - v_2 - \cdots - v_k$ whose endpoints (v_0 and v_k) are in V and whose internal vertices (v_i for $1 \leq i \leq k-1$) are not in V. We allow that v_0 be equal to v_k, in which case the path would reduce to a cycle. Adding the ear to H creates a new graph on $V \cup \{v_1, \cdots, v_{k-1}\}$. The trivial case when $k = 1$ (a 'trivial' ear) simply means adding an edge to H. An ear is called *odd* if k is odd, and even otherwise; for example, a trivial ear is odd.

   (a)Let G be a graph that can be constructed by starting from an odd cycle and repeatedly adding odd ears. Prove that G is factor-critical.

   (b)Prove the converse that any factor-critical graph can be built by starting from an odd cycle and repeatedly adding odd ears.

3. Linear Programming and Polyhedral Combinatorics

Summary of what was seen in the introductory lectures on linear programming and polyhedral combinatorics.

Definition 3.1 *A halfspace in $\mathbb{R}^n$ is a set of the form $\{x \in \mathbb{R}^n : a^T x \leq b\}$ for some vector $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$.*

Definition 3.2 *A polyhedron is the intersection of finitely many halfspaces: $P = \{x \in \mathbb{R}^n : Ax \leq b\}$.*

Definition 3.3 *A polytope is a bounded polyhedron.*

Definition 3.4 *If P is a polyhedron in $\mathbb{R}^n$, the projection $P_k \subseteq \mathbb{R}^{n-1}$ of P is defined as $\{y = (x_1, x_2, \cdots, x_{k-1}, x_{k+1}, \cdots, x_n) : x \in P \text{ for some } x_k \in \mathbb{R}\}$.*

This is a special case of a projection onto a linear space (here, we consider only coordinate projection). By repeatedly projecting, we can eliminate any subset of coordinates.

We claim that P_k is also a polyhedron and this can be proved by giving an explicit description of P_k in terms of linear inequalities. For this purpose, one uses *Fourier-Motzkin elimination*. Let $P = \{x : Ax \leq b\}$ and let

- $S_+ = \{i : a_{ik} > 0\}$,
- $S_- = \{i : a_{ik} < 0\}$,
- $S_0 = \{i : a_{ik} = 0\}$.

Clearly, any element in P_k must satisfy the inequality $a_i^T x \leq b_i$ for all $i \in S_0$ (these inequalities do not involve x_k). Similarly, we can take a linear combination of an inequality in S_+ and one in S_- to eliminate the coefficient of x_k. This shows that the inequalities:

$$a_{ik}\left(\sum_j a_{lj}x_j\right) - a_{lk}\left(\sum_j a_{ij}x_j\right) \leq a_{ik}b_l - a_{lk}b_i \tag{1}$$

for $i \in S_+$ and $l \in S_-$ are satisfied by all elements of P_k. Conversely, for any vector $(x_1, x_2, \cdots, x_{k-1}, x_{k+1}, \cdots, x_n)$ satisfying (1) for all $i \in S_+$ and $l \in S_-$ and also

$$a_i^T x \leq b_i \text{ for all } i \in S_0 \tag{2}$$

we can find a value of x_k such that the resulting x belongs to P (by looking at the bounds on x_k that each constraint imposes, and showing that the largest lower bound is smaller than the smallest upper bound). This shows that P_k is described by (1) and (2), and therefore is a polyhedron.

Definition 3.5 *Given points $a^{(1)}, a^{(2)}, \cdots, a^{(k)} \in \mathbb{R}^n$,*

- *a* **linear** *combination is $\sum_i \lambda_i a^{(i)}$ where $\lambda_i \in \mathbb{R}$ for all i,*
- *an* **affine** *combination is $\sum_i \lambda_i a^{(i)}$ where $\lambda_i \in \mathbb{R}$ and $\sum_i \lambda_i = 1$,*
- *a* **conical** *combination is $\sum_i \lambda_i a^{(i)}$ where $\lambda_i \geq 0$ for all i,*
- *a* **convex** *combination is $\sum_i \lambda_i a^{(i)}$ where $\lambda_i \geq 0$ for all i and $\sum_i \lambda_i = 1$.*

The set of all linear combinations of elements of S is called the linear hull of S and denoted by $\text{lin}(S)$. Similarly, by replacing *linear* by *affine, conical or convex*, we define the affine hull, $\text{aff}(S)$, the conic hull, $\text{cone}(S)$ and the convex hull, $\text{conv}(S)$. We can give an equivalent definition of a polytope.

Definition 3.6 *A polytope is the convex hull of a finite set of points.*

The fact that Definition 3.6 implies Definition 3.3 can be seen as follows. Take P be the convex hull of a finite set $\{a^{(k)}\}_{k\in[m]}$ of points. To show that P can be described as the intersection of a finite number of hyperplanes, we can apply Fourier-Motzkin elimination repeatedly on

$$x - \sum_k \lambda_k a^{(k)} = 0$$

$$\sum_k \lambda_k = 1$$

$$\lambda_k \geq 0$$

to eliminate all variables λ_k and keep only the variables x. Furthermore, P is bounded since for any $x \in P$, we have

$$||x|| = ||\sum_k \lambda_k a^{(k)}|| \leq \sum_k \lambda_k ||a^{(k)}|| \leq \max_k ||a^{(k)}||.$$

The converse will be proved later in these notes.

3.1 Solvability of System of Inequalities

In linear algebra, we saw that, for $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $Ax = b$ has no solution $x \in \mathbb{R}^n$ if and only if there exists a $y \in \mathbb{R}^m$ with $A^T y = 0$ and $b^T y \neq 0$ (in 18.06 notation/terminology, this is equivalent to saying that the column space $C(A)$ is orthogonal to the left null space $N(A^T)$).

One can state a similar *Theorem of the Alternatives* for systems of linear inequalities.

Theorem 3.1 (Theorem of the Alternatives) *$Ax \leq b$ has no solution $x \in \mathbb{R}^n$ if and only if there exists $y \in \mathbb{R}^m$ such that $y \geq 0$, $A^T y = 0$ and $b^T y < 0$.*

One can easily show that both systems indeed cannot have a solution since otherwise $0 > b^T y = y^T b \geq y^T Ax = 0^T x = 0$. For the other direction, one takes the insolvable system $Ax \leq b$ and use Fourier-Motzkin elimination repeatedly to eliminate all variables and thus obtain an inequality of the form $0^T x \leq c$ where $c < 0$. In the process one has derived a vector y with the desired properties (as Fourier-Motzkin only performs nonnegative combinations of linear inequalities).

Another version of the above theorem is Farkas' lemma:

Lemma 3.2 *$Ax = b$, $x \geq 0$ has no solution if and only if there exists y with $A^T y \geq 0$ and $b^T y < 0$.*

Exercise 3-1. Prove Farkas' lemma from the Theorem of the Alternatives.

3.2 Linear Programming Basics

A linear program (LP) is the problem of minimizing or maximizing a linear function over a polyhedron:

$$\begin{aligned} & \text{Max} && c^T x \\ & \text{subject to:} && \\ (P) \qquad & && Ax \leq b, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and the variables x are in $\mathbb{R}^n$. Any x satisfying $Ax \leq b$ is said to be *feasible*. If no x satisfies $Ax \leq b$, we say that the linear program is *infeasible*, and its optimum value is $-\infty$ (as we are maximizing over an empty set). If the objective function value of the linear program can be made arbitrarily large, we say that the linear program is *unbounded* and its optimum value is $+\infty$; otherwise it is *bounded*. If it is neither infeasible, nor unbounded, then its optimum value is finite.

Other equivalent forms involve equalities as well, or nonnegative constraints $x \geq 0$. One version that is often considered when discussing algorithms for linear programming (especially the simplex algorithm) is $\min\{c^T x : Ax = b, x \geq 0\}$.

Another linear program, *dual* to (P), plays a crucial role:

$$\begin{aligned} & \text{Min} && b^T y \\ & \text{subject to:} && \\ (D) \qquad & && A^T y = c \\ & && y \geq 0. \end{aligned}$$

(D) is the dual and (P) is the *primal*. The terminology for the dual is similar. If (D) has no feasible solution, it is said to be *infeasible* and its optimum value is $+\infty$ (as we are minimizing over an empty set). If (D) is unbounded (i.e. its value can be made arbitrarily negative) then its optimum value is $-\infty$.

The primal and dual spaces should not be confused. If A is $m \times n$ then we have n primal variables and m dual variables.

Weak duality is clear: For any feasible solutions x and y to (P) and (D), we have that $c^T x \leq b^T y$. Indeed, $c^T x = y^T A x \leq b^T y$. The dual was precisely built to get an upper bound on the value of any primal solution. For example, to get the inequality $y^T A x \leq b^T y$, we need that $y \geq 0$ since we know that $Ax \leq b$. In particular, weak duality implies that if the primal is unbounded then the dual must be infeasible.

Strong duality is the most important result in linear programming; it says that we can prove the optimality of a primal solution x by exhibiting an optimum dual solution y.

Theorem 3.3 (Strong Duality) *Assume that (P) and (D) are feasible, and let z^* be the optimum value of the primal and w^* the optimum value of the dual. Then $z^* = w^*$.*

One proof of strong duality is obtained by writing a big system of inequalities in x and y which says that (i) x is primal feasible, (ii) y is dual feasible and (iii) $c^T x \geq b^T y$. Then use the Theorem of the Alternatives to show that the infeasibility of this system of inequalities would contradict the feasibility of either (P) or (D).

Proof: Let x^* be a feasible solution to the primal, and y^* be a feasible solution to the dual. The proof is by contradiction. Because of weak duality, this means that there are no solution $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ such that

$$\begin{cases} Ax & & \leq b \\ & A^T y & = c \\ & -Iy & \leq 0 \\ -c^T x & +b^T y & \leq 0 \end{cases}$$

By a variant of the Theorem of the Alternatives or Farkas' lemma (for the case when we have a combination of inequalities and equalities), we derive that there must exist $s \in \mathbb{R}^m$, $t \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $v \in \mathbb{R}$ such that:

$$\begin{aligned} & s \geq 0 \\ & u \geq 0 \\ & v \geq 0 \\ & A^T s - vc = 0 \\ & At - u + vb = 0 \\ & b^T s + c^T t < 0. \end{aligned}$$

We distinguish two cases.

Case 1: $v = 0$**.** Then s satisfies $s \geq 0$ and $A^T s = 0$. This means that, for any $\alpha \geq 0$, $y^* + \alpha s$ is feasible for the dual. Similarly, $At = u \geq 0$ and therefore, for any $\alpha \geq 0$, we have that $x^* - \alpha t$ is primal feasible. By weak duality, this means that, for any $\alpha \geq 0$, we have

$$c^T(x^* - \alpha t) \leq b^T(y^* + \alpha s)$$

or

$$c^T x^* - b^T y^* \leq \alpha(b^T s + c^T t).$$

The right-hand-side tend to $-\infty$ as α tends to ∞, and this is a contradiction as the left-hand-side is fixed.

Case 2: $v > 0$**.** By dividing throughout by v (and renaming all the variables), we get that there exists $s \geq 0$, $u \geq 0$ with

$$\begin{aligned} A^T s &= c \\ At - u &= -b \\ b^T s + c^T t &< 0. \end{aligned}$$

This means that s is dual feasible and $-t$ is primal feasible, and therefore by weak duality $c^T(-t) \leq b^T s$ contradicting $b^T s + c^T t < 0$. △

Exercise 3-2. Show that the dual of the dual is the primal.

Exercise 3-3. Show that we only need either the primal or the dual to be feasible for strong duality to hold. More precisely, if the primal is feasible but the dual is infeasible, prove that the primal will be unbounded, implying that $z^* = w^* = +\infty$.

Looking at $c^T x = y^T A x \leq b^T y$, we observe that to get equality between $c^T x$ and $b^T y$, we need *complementary slackness*:

Theorem 3.4 (Complementary Slackness) *If x is feasible in (P) and y is feasible in (D) then x is optimum in (P) and y is optimum in (D) if and only if for all i either $y_i = 0$ or $\sum_j a_{ij} x_j = b_i$ (or both).*

Linear programs can be solved using the simplex method; this is not going to be explained in these notes. No variant of the simplex method is known to provably run in polynomial time, but there are other polynomial-time algorithms for linear programming, namely the ellipsoid algorithm and the class of interior-point algorithms.

3.3 Faces of Polyhedra

Definition 3.7 *$\{a^{(i)} \in \mathbb{R}^n : i \in K\}$ are linearly independent if $\sum_i \lambda_i a^{(i)} = 0$ implies that $\lambda_i = 0$ for all $i \in K$.*

Definition 3.8 *$\{a^{(i)} \in \mathbb{R}^n : i \in K\}$ are affinely independent if $\sum_i \lambda_i a^{(i)} = 0$ and $\sum_i \lambda_i = 0$ together imply that $\lambda_i = 0$ for all $i \in K$.*

Observe that $\{a^{(i)} \in \mathbb{R}^n : i \in K\}$ are *affinely* independent if and only if

$$\left\{ \begin{bmatrix} a^{(i)} \\ 1 \end{bmatrix} \in \mathbb{R}^{n+1} : i \in K \right\}$$

are *linearly* independent.

Definition 3.9 *The dimension,* $\dim(P)$*, of a polyhedron* P *is the maximum number of affinely independent points in* P *minus 1.*

(This is the same notion as the dimension of the affine hull $\text{aff}(S)$.) The dimension can be -1 (if P is empty), 0 (when P consists of a single point), 1 (when P is a line segment), and up to n when P affinely spans $\mathbb{R}^n$. In the latter case, we say that P is *full-dimensional*. The dimension of a cube in $\mathbb{R}^3$ is 3, and so is the dimension of $\mathbb{R}^3$ itself (which is a trivial polyhedron).

Definition 3.10 $\alpha^T x \leq \beta$ *is a valid inequality for* P *if* $\alpha^T x \leq \beta$ *for all* $x \in P$*.*

Observe that for an inequality to be valid for $\text{conv}(S)$ we only need to make sure that it is satisfied by all elements of S, as this will imply that the inequality is also satisfied by points in $\text{conv}(S) \setminus S$. This observation will be important when dealing with convex hulls of combinatorial objects such as matchings or spanning trees.

Definition 3.11 *A face of a polyhedron* P *is* $\{x \in P : \alpha^T x = \beta\}$ *where* $\alpha^T x \leq \beta$ *is some valid inequality of* P*.*

By definition, all faces are polyhedra. The empty face (of dimension -1) is *trivial*, and so is the entire polyhedron P (which corresponds to the valid inequality $0^T x \leq 0$). Non-trivial are those whose dimension is between 0 and $\dim(P) - 1$. Faces of dimension 0 are called *extreme points* or *vertices*, faces of dimension 1 are called *edges*, and faces of dimension $\dim(P) - 1$ are called *facets*. Sometimes, one uses *ridges* for faces of dimension $\dim(P) - 2$.

Exercise 3-4. List all 28 faces of the cube $P = \{x \in \mathbb{R}^3 : 0 \leq x_i \leq 1 \text{ for } i = 1, 2, 3\}$.

Although there are infinitely many valid inequalities, there are only finitely many faces.

Theorem 3.5 *Let* $A \in \mathbb{R}^{m \times n}$*. Then any non-empty face of* $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ *corresponds to the set of solutions to*

$$\sum_j a_{ij} x_j = b_i \text{ for all } i \in I$$

$$\sum_j a_{ij} x_j \leq b_i \text{ for all } i \notin I,$$

for some set $I \subseteq \{1, \cdots, m\}$*. Therefore, the number of non-empty faces of* P *is at most* 2^m*.*

Proof: Consider any valid inequality $\alpha^T x \leq \beta$. Suppose the corresponding face F is non-empty. Thus F are all optimum solutions to

$$\begin{array}{ll} & \text{Max} \quad \alpha^T x \\ & \text{subject to:} \\ (P) & \qquad Ax \leq b. \end{array}$$

Choose an optimum solution y^* to the dual LP. By complementary slackness, the face F is defined by those elements x of P such that $a_i^T x = b_i$ for $i \in I = \{i : y_i^* > 0\}$. Thus F is defined by

$$\sum_j a_{ij} x_j = b_i \text{ for all } i \in I$$

$$\sum_j a_{ij} x_j \leq b_i \text{ for all } i \notin I.$$

As there are 2^m possibilities for I, there are at most 2^m non-empty faces. $\triangle$

The number of faces given in Theorem 3.5 is tight for polyhedra (see exercise below), but can be considerably improved for polytopes in the so-called *upper bound theorem* (which is not given in these notes).

Exercise 3-5. Let $P = \{x \in \mathbb{R}^n : x_i \geq 0 \text{ for } i = 1, \cdots, n\}$. Show that P has $2^n + 1$ faces. How many faces of dimension k does P have?

For extreme points (faces of dimension 0), the characterization is even stronger (we do not need the inequalities):

Theorem 3.6 *Let x^* be an extreme point for $P = \{x : Ax \leq b\}$. Then there exists I such that x^* is the unique solution to*

$$\sum_j a_{ij} x_j = b_i \text{ for all } i \in I.$$

Proof: Given an extreme point x^*, define $I = \{i : \sum_j a_{ij} x_j^* = b_i\}$. This means that for $i \notin I$, we have $\sum_j a_{ij} x_j^* < b_i$.

From Theorem 3.5, we know that x^* is uniquely defined by

$$\sum_j a_{ij} x_j = b_i \text{ for all } i \in I \tag{3}$$

$$\sum_j a_{ij} x_j \leq b_i \text{ for all } i \notin I. \tag{4}$$

Now suppose there exists another solution $\hat{x}$ *when* we consider only the equalities for $i \in I$. Then because of $\sum_j a_{ij} x_j^* < b_i$, we get that $(1 - \epsilon)x^* + \epsilon\hat{x}$ also satisfies (3) and (4) for ϵ sufficiently small. A contradiction (as the face was supposed to contain a single point). $\triangle$

If P is given as $\{x : Ax = b, x \geq 0\}$ (as is often the case), the theorem still applies (as we still have a system of inequalities). In this case, the theorem says that every extreme point x^* can be obtained by setting some of the variables to 0, and solving for the unique solution to the resulting system of equalities. Without loss of generality, we can remove from $Ax = b$ equalities that are redundant; this means that we can assume that A has full row rank ($\text{rank}(A) = m$ for $A \in \mathbb{R}^{m \times n}$). Letting N denote the indices of the *non-basic* variables that we set of 0 and B denote the remaining indices (of the so-called *basic* variables), we can partition x^* into x^*_B and x^*_N (corresponding to these two sets of variables) and rewrite $Ax = b$ as $A_B x_B + A_N x_N = b$, where A_B and A_N are the restrictions of A to the indices in B and N respectively. The theorem says that x^* is the unique solution to $A_B x_B + A_N x_N = 0$ and $x_N = 0$, which means $x^*_N = 0$ and $A_B x^*_B = b$. This latter system must have a unique solution, which means that A_B must have full column rank ($\text{rank}(A_B) = |B|$). As A itself has rank m, we have that $|B| \leq m$ and we can augment B to include indices of N such that the resulting B satisfies (i) $|B| = m$ and (ii) A_B is a $m \times m$ invertible matrix (and thus there is still a unique solution to $A_B x_B = b$). In linear programming terminology, a *basic feasible solution* or *bfs* of $\{x : Ax = b, x \geq 0\}$ is obtained by choosing a set $|B| = m$ of indices with A_B invertible and letting $x_B = A_B^{-1} b$ and $x_N = 0$ where N are the indices not in B. We have thus shown that all extreme points are bfs, and vice versa. Observe that two different bases B may lead to the same extreme point, as there might be many ways of extending A_B into a $m \times m$ invertible matrix in the discussion above.

One consequence we could derive from Theorem 3.5 is:

Corollary 3.7 *The maximal (inclusion-wise) non-trivial faces of a non-empty polyhedron P are the facets.*

For the vertices, one needs one additional condition:

Corollary 3.8 *If $\text{rank}(A) = n$ (full column rank) then the minimal (inclusion-wise) non-trivial faces of a non-empty polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ are the vertices.*

Exercise 3-7 shows that the rank condition is necessary.

This means that, if a linear program $\max\{c^T x : x \in P\}$ with $P = \{x : Ax \leq b\}$ is feasible, bounded and $\text{rank}(A) = n$, then there exists an optimal solution which is a vertex of P (indeed, the set of all optimal solutions defines a face — the optimal face — and if this face is not itself a vertex of P, it must contain vertices of P).

We now prove Corollary 3.8.

Proof: Let F be a minimal (inclusion-wise) non-trivial face of P. This means that we have a set I such that

$$F = \{x : \begin{array}{ll} a_i^T x = b_i & \forall i \in I \\ a_j^T x \leq b_j & \forall j \notin I\} \end{array}$$

and adding any element to I makes this set empty. Consider two cases. Either $F = \{x \in \mathbb{R}^n : a_i^T x = b_i \text{ for } i \in I\}$ or not. In the first case, it means that for every $j \notin I$ we have $a_j \in \text{lin}(\{a_i : i \in I\})$ (otherwise there would be a solution x to $a_i^T x = b_i$ for all $i \in I$ and

$a_j^T x = b_j + 1$ and hence not in F) and therefore since $\text{rank}(A) = n$ we have that the system $a_i^T x = b_i$ for all $i \in I$ has a unique solution and thus F is a vertex.

On the other hand, if $F \neq \{x \in \mathbb{R}^n : a_i^T x = b_i \text{ for } i \in I\}$ then let $j \notin I$ such that there exists $\tilde{x}$ with

$$\begin{aligned} a_i^T \tilde{x} &= b_i \quad i \in I \\ a_j^T \tilde{x} &> b_j. \end{aligned}$$

Since F is not trivial, there exists $\hat{x} \in F$. In particular, $\hat{x}$ satisfies

$$\begin{aligned} a_i^T \hat{x} &= b_i \quad i \in I \\ a_j^T \hat{x} &\leq b_j. \end{aligned}$$

Consider a convex combination $x' = \lambda \tilde{x} + (1 - \lambda)\hat{x}$. Consider the largest λ such that x' is in P. This is well-defined as $\lambda = 0$ gives a point in P while it is not for $\lambda = 1$. The corresponding x' satisfies $a_i^T x' = b_i$ for $i \in I \cup \{k\}$ for some k (possibly j), contradicting the maximality of I. △

We now go back to the equivalence between Definitions 3.3 and 3.6 and claim that we can show that Definition 3.3 implies Definition 3.6.

Theorem 3.9 *If $P = \{x : Ax \leq b\}$ is bounded then $P = \text{conv}(X)$ where X is the set of extreme points of P.*

This is a nice exercise using the Theorem of the Alternatives.

Proof: Since $X \subseteq P$, we have $\text{conv}(X) \subseteq P$. Assume, by contradiction, that we do not have equality. Then there must exist $\tilde{x} \in P \setminus \text{conv}(X)$. The fact that $\tilde{x} \notin \text{conv}(X)$ means that there is no solution to:

$$\begin{cases} \sum_{v \in X} \lambda_v v = \tilde{x} \\ \sum_{v \in X} \lambda_v = 1 \\ \lambda_v \geq 0 \qquad\qquad v \in X. \end{cases}$$

By the Theorem of the alternatives, this implies that $\exists c \in \mathbb{R}^n, t \in \mathbb{R}$:

$$\begin{cases} t + \sum_{j=1}^n c_j v_j \geq 0 \quad \forall v \in X \\ t + \sum_{j=1}^n c_j \tilde{x}_j < 0. \end{cases}$$

Since P is bounded, $\min\{c^T x : x \in P\}$ is finite (say equal to z^*), and the face induced by $c^T x \geq z^*$ is non-empty but does not contain any vertex (as all vertices are dominated by $\tilde{x}$ by the above inequalities). This is a contradiction with Corollary 3.8. Observe, indeed, that Corollary 3.8 applies. If $\text{rank}(A) < n$ there woule exists $y \neq 0$ with $Ay = 0$ and this would contradict the boundedness of P (as we could go infinitely in the direction of y). △

When describing a polyhedron P in terms of linear inequalities, the only inequalities that are needed are the ones that define facets of P. This is stated in the next few theorems. We say that an inequality in the system $Ax \leq b$ is *redundant* if the corresponding polyhedron is unchanged by removing the inequality. For $P = \{x : Ax \leq b\}$, we let $I_=$ denote the indices i such that $a_i^T x = b_i$ for all $x \in P$, and $I_<$ the remaining ones (i.e. those for which there exists $x \in P$ with $a_i^T x < b_i$).

This theorem shows that facets are sufficient:

Theorem 3.10 *If the face associated with $a_i^T x \leq b_i$ for $i \in I_<$ is not a facet then the inequality is redundant.*

And this one shows that facets are necessary:

Theorem 3.11 *If F is a facet of P then there must exists $i \in I_<$ such that the face induced by $a_i^T x \leq b_i$ is precisely F.*

In a *minimal* description of P, we must have a set of *linearly independent equalities* together with precisely one inequality for each facet of P.

Exercises

Exercise 3-6. Prove Corollary 3.7.

Exercise 3-7. Show that if $\text{rank}(A) < n$ then $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ has no vertices.

Exercise 3-8. Suppose $P = \{x \in \mathbb{R}^n : Ax \leq b, Cx \leq d\}$. Show that the set of vertices of $Q = \{x \in \mathbb{R}^n : Ax \leq b, Cx = d\}$ is a subset of the set of vertices of P.

(In particular, this means that if the vertices of P all belong to $\{0,1\}^n$, then so do the vertices of Q.)

Exercise 3-9. Given two extreme points a and b of a polyhedron P, we say that they are *adjacent* if the line segment between them forms an edge (i.e. a face of dimension 1) of the polyhedron P. This can be rephrased by saying that a and b are adjacent on P if and only if there exists a cost function c such that a and b are the only two extreme points of P minimizing $c^T x$ over P.

Consider the polyhedron (polytope) P defined as the convex hull of all perfect matchings in a (not necessarily bipartite) graph G. Give a necessary and sufficient condition for two matchings M_1 and M_2 to be adjacent on this polyhedron (hint: think about $M_1 \triangle M_2 = (M_1 \setminus M_2) \cup (M_2 \setminus M_1)$) and prove that your condition is necessary and sufficient.)

Exercise 3-10. Show that two vertices u and v of a polyhedron P are adjacent if and only there is a unique way to express their midpoint ($\frac{1}{2}(u+v)$) as a convex combination of vertices of P.

3.4 Polyhedral Combinatorics

In one sentence, polyhedral combinatorics deals with the study of polyhedra or polytopes associated with discrete sets arising from combinatorial optimization problems (such as matchings for example). If we have a discrete set X (say the incidence vectors of matchings in a graph, or the set of incidence vectors of spanning trees of a graph, or the set of incidence vectors of *stable* sets[1] in a graph), we can consider $\text{conv}(X)$ and attempt to describe it in terms

[1] A set S of vertices in a graph $G = (V, E)$ is stable if there are no edges between any two vertices of S.

of linear inequalities. This is useful in order to apply the machinery of linear programming. However, in some (most) cases, it is actually hard to describe the set of all inequalities defining $\text{conv}(X)$; this occurs whenever optimizing over X is hard and this statement can be made precise in the setting of computational complexity. For matchings, or spanning trees, and several other structures (for which the corresponding optimization problem is polynomially solvable), we will be able to describe their convex hull in terms of linear inequalities.

Given a set X and a proposed system of inequalities $P = \{x : Ax \leq b\}$, it is usually easy to check whether $\text{conv}(X) \subseteq P$. Indeed, for this, we only need to check that every member of X satisfies every inequality in the description of P. The reverse inclusion is more difficult. Here are 3 general techniques to prove that $P \subseteq \text{conv}(X)$ (if it is true!) (once we know that $\text{conv}(X) \subseteq P$).

1. **Algorithmically.** This involves linear programming duality. This is what we did in the notes about the assignment problem (minimum weight matchings in bipartite graphs). In general, consider any cost function c and consider the combinatorial optimization problem of maximizing $c^T x$ over $x \in X$. We know that:

$$\begin{aligned} \max\{c^T x : x \in X\} &= \max\{c^T x : x \in \text{conv}(X)\} \\ &\leq \max\{c^T x : Ax \leq b\} \\ &= \min\{b^T y : A^T y = c, y \geq 0\}, \end{aligned}$$

   the last equality coming from strong duality. If we can exhibit a solution $x \in X$ (say the incidence vector of a perfect matching in the assignment problem) and a dual feasible solution y (values u_i, v_j in the assignment problem) such that $c^T x = b^T y$ we will have shown that we have equality throughout, and if this is true for *any* cost function c, this implies that $P = \text{conv}(X)$.

   This is usually the most involved approach but also the one that works most often.

2. **Focusing on extreme points.** Show first that $P = \{x : Ax \leq b\}$ is bounded (thus a polytope) and then study its extreme points. If we can show that every extreme point of P is in X then we would be done since $P = \text{conv}(ext(P)) \subseteq \text{conv}(X)$, where $ext(P)$ denotes the extreme points of P (see Theorem 3.9). The assumption that P is bounded is needed to show that indeed $P = \text{conv}(ext(P))$ (not true if P is unbounded).

   In the case of the convex hull of bipartite matchings, this can be done easily and this leads to the notion of *totally unimodular* Matrices (TU), see the next section.

3. **Focusing on the facets of** $\text{conv}(X)$**.** This leads usually to the shortest and cleanest proofs. Suppose that our proposed P is of the form $\{x \in \mathbb{R}^n : Ax \leq b, Cx = d\}$. We have already argued that $\text{conv}(X) \subseteq P$ and we want to show that $P \subseteq \text{conv}(X)$.

   First we need to show that we are not missing any equality. This can be done for example by showing that $\dim(\text{conv}(X)) = \dim(P)$. We already know that $\dim(\text{conv}(X)) \leq \dim(P)$ (as $\text{conv}(X) \subseteq P$), and so we need to argue that $\dim(conv(X)) \geq \dim(P)$.

This means showing that if there are $n-d$ linearly independent rows in C we can find $d+1$ affinely independent points in X.

Then we need to show that we are not missing a valid inequality that induces a *facet* of $\text{conv}(X)$. Consider any valid inequality $\alpha^T x \leq \beta$ for $\text{conv}(X)$ with $\alpha \neq 0$. We can assume that α is any vector in $\mathbb{R}^n \setminus \{0\}$ and that $\beta = \max\{\alpha^T x : x \in \text{conv}(X)\}$. The face of $\text{conv}(X)$ this inequality defines is $F = \text{conv}(\{x \in X : \alpha^T x = \beta\})$. Assume that this is a non-trivial face; this will happen precisely when α is not in the row space of C. We need to make sure that if F is a facet then we have in our description of P an inequality representing it. What we will show is that if F is non-trivial then we can find an inequality $a_i^T x \leq b_i$ in our description of P such that (i) $F \subseteq \{x : a_i^T x = b_i\}$ and (ii) $a_i^T x \leq b_i$ defines a non-trivial face of P (this second condition is not needed if P is full-dimensional), or simply that every optimum solution to $\max\{\alpha^T x : x \in X\}$ satisfies $a_i^T x = b_i$, and that this inequality is not satisfied by all points in P. This means that if F was a facet, by maximality, we have a representative of F in our description.

This is a very simple and powerful technique, and this is best illustrated on an example.

Example. Let $X = \{(\sigma(1), \sigma(2), \cdots, \sigma(n)) : \sigma \text{ is a permutation of } \{1, 2, \cdots, n\}\}$. We claim that

$$\text{conv}(X) = \{x \in \mathbb{R}^n : \begin{array}{ll} \sum_{i=1}^n x_i = \binom{n+1}{2} & \\ \sum_{i \in S} x_i \geq \binom{|S|+1}{2} & S \subset \{1, \cdots, n\}\}. \end{array}$$

This is known as the *permutahedron*.

Here $\text{conv}(X)$ is not full-dimensional; we only need to show that we are not missing any facets and any equality in the description of $\text{conv}(P)$. For the equalities, this can be seen easily as it is easy to exhibit n affinely independent permutations in X. For the facets, suppose that $\alpha^T x \leq \beta$ defines a non-trivial facet F of $\text{conv}(X)$. Consider maximizing $\alpha^T x$ over all permutations x. Let $S = \arg\min\{\alpha_i\}$; by our assumption that F is non-trivial we have that $S \neq \{1, 2, \cdots, n\}$ (otherwise, we would have the equality $\sum_{i=1}^n x_i = \binom{n+1}{2}$). Moreover, it is easy to see (by an exchange argument) that any permutation σ whose incidence vector x maximizes $\alpha^T x$ will need to satisfy $\sigma(i) \in \{1, 2, \cdots, |S|\}$ for $i \in S$, in other words, it will satisfy the inequality $\sum_{i \in S} x_i \geq \binom{|S|+1}{2}$ at equality (and this is a non-trivial face as there exist permutations that do not satisfy it at equality). Hence, F is contained in a non-trivial face corresponding to an inequality in our description, and hence our description contains inequalities for all facets. This is what we needed to prove. That's it!

Exercises

Exercise 3-11. Consider the set $X = \{(\sigma(1), \sigma(2), \cdots, \sigma(n)) : \sigma \text{ is a permutation of } \{1, 2 \cdots, n\}\}$. Show that $\dim(\text{conv}(X)) = n - 1$. (To show that $\dim(\text{conv}(X)) \geq n - 1$, ex-

hibit n affinely independent permutations σ (and prove that they are affinely independent).)

Exercise 3-12. A *stable set* S (sometimes, it is called also an independent set) in a graph $G = (V, E)$ is a set of vertices such that there are no edges between any two vertices in S. If we let P denote the convex hull of all (incidence vectors of) stable sets of $G = (V, E)$, it is clear that $x_i + x_j \leq 1$ for any edge $(i, j) \in E$ is a valid inequality for P.

1. Give a graph G for which P is *not* equal to

$$\{x \in \mathbb{R}^{|V|} : \quad x_i + x_j \leq 1 \quad \text{for all } (i,j) \in E \\ x_i \geq 0 \quad \text{for all } i \in V\}$$

2. Show that if the graph G is bipartite then P equals

$$\{x \in \mathbb{R}^{|V|} : \quad x_i + x_j \leq 1 \quad \text{for all } (i,j) \in E \\ x_i \geq 0 \quad \text{for all } i \in V\}.$$

Exercise 3-13. Let $e_k \in \mathbb{R}^n$ $(k = 0, \ldots, n-1)$ be a vector with the first k entries being 1, and the following $n - k$ entries being -1. Let $S = \{e_0, e_1, \ldots, e_{n-1}, -e_0, -e_1, \ldots, -e_{n-1}\}$, i.e. S consists of all vectors consisting of $+1$ followed by -1 or vice versa. In this problem set, you will study $\text{conv}(S)$.

1. Consider any vector $a \in \{-1, 0, 1\}^n$ such that (i) $\sum_{i=1}^n a_i = 1$ and (ii) for all $j = 1, \ldots, n-1$, we have $0 \leq \sum_{i=1}^j a_i \leq 1$. (For example, for $n = 5$, the vector $(1, 0, -1, 1, 0)$ satisfies these conditions.) Show that $\sum_{i=1}^n a_i x_i \leq 1$ and $\sum_{i=1}^n a_i x_i \geq -1$ are valid inequalities for $\text{conv}(S)$.
2. How many such inequalities are there?
3. Show that any such inequality defines a facet of $\text{conv}(S)$.

   (This can be done in several ways. Here is one approach, but you are welcome to use any other one as well. First show that either e_k or $-e_k$ satisfies this inequality at equality, for any k. Then show that the resulting set of vectors on the hyperplane are affinely independent (or uniquely identifies it).)
4. Show that the above inequalities define the entire convex hull of S.

   (Again this can be done in several ways. One possibility is to consider the 3rd technique described above.)

3.5 Total unimodularity

Definition 3.12 *A matrix A is totally unimodular (TU) if every square submatrix of A has determinant $-1, 0$ or $+1$.*

The importance of total unimodularity stems from the following theorem. This theorem gives a subclass of integer programs which are easily solved. A polyhedron P is said to be *integral* if all its vertices or extreme points are integral (belong to $\mathbb{Z}^n$).

Theorem 3.12 *Let A be a totally unimodular matrix. Then, for any integral right-hand-side b, the polyhedron*

$$P = \{x : Ax \leq b, x \geq 0\}$$

is integral.

Before we prove this result, two remarks can be made. First, the proof below will in fact show that the same result holds for the polyhedrons $\{x : Ax \geq b, x \geq 0\}$ or $\{x : Ax = b, x \geq 0\}$. In the latter case, though, a slightly weaker condition than totally unimodularity is sufficient to prove the result. Secondly, in the above theorem, one can prove the converse as well: If $P = \{x : Ax \leq b, x \geq 0\}$ is integral for all integral b then A must be totally unimodular (this is not true though, if we consider for example $\{x : Ax = b, x \geq 0\}$).

Proof: Adding slacks, we get the polyhedron $Q = \{(x, s) : Ax + Is = b, x \geq 0, s \geq 0\}$. One can easily show (see exercise below) that P is integral iff Q is integral.

Consider now any bfs of Q. The basis B consists of some columns of A as well as some columns of the identity matrix I. Since the columns of I have only one nonzero entry per column, namely a one, we can expand the determinant of A_B along these entries and derive that, in absolute values, the determinant of A_B is equal to the determinant of some square submatrix of A. By definition of totally unimodularity, this implies that the determinant of A_B must belong to $\{-1, 0, 1\}$. By definition of a basis, it cannot be equal to 0. Hence, it must be equal to ± 1.

We now prove that the bfs must be integral. The non-basic variables, by definition, must have value zero. The vector of basic variables, on the other hand, is equal to $A_B^{-1}b$. From linear algebra, A_B^{-1} can be expressed as

$$\frac{1}{\det A_B} A_B^{adj}$$

where A_B^{adj} is the adjoint (or adjugate) matrix of A_B and consists of subdeterminants of A_B. Hence, both b and A_B^{adj} are integral which implies that $A_B^{-1}b$ is integral since $|\det A_B| = 1$. This proves the integrality of the bfs. △

Exercise 3-14. Let $P = \{x : Ax \leq b, x \geq 0\}$ and let $Q = \{(x, s) : Ax + Is = b, x \geq 0, s \geq 0\}$. Show that x is an extreme point of P iff $(x, b - Ax)$ is an extreme point of Q. Conclude that whenever A and b have only integral entries, P is integral iff Q is integral.

In the case of the bipartite matching problem, the constraint matrix A has a very special structure and we show below that it is totally unimodular. This together with Theorem 3.12 proves Theorem 1.6 from the notes on the bipartite matching problem. First, let us

restate the setting. Suppose that the bipartition of our bipartite graph is (U, V) (to avoid any confusion with the matrix A or the basis B). Consider

$$\begin{aligned} P &= \{x: \sum_j x_{ij} = 1 && i \in U \\ &\quad \sum_i x_{ij} = 1 && j \in V \\ &\quad x_{ij} \geq 0 && i \in U, j \in V\} \\ &= \{x: Ax = b, x \geq 0\}. \end{aligned}$$

Theorem 3.13 *The matrix A is totally unimodular.*

The way we defined the matrix A corresponds to a *complete* bipartite graph. If we were to consider any bipartite graph then we would simply consider a submatrix of A, which is also totally unimodular by definition.

Proof: Consider any square submatrix T of A. We consider three cases. First, if T has a column or a row with all entries equal to zero then the determinant is zero. Secondly, if there exists a column or a row of T with only one $+1$ then by expanding the determinant along that $+1$, we can consider a smaller sized matrix T. The last case is when T has at least two nonzero entries per column (and per row). Given the special structure of A, there must in fact be *exactly* two nonzero entries per column. By adding up the rows of T corresponding to the vertices of U and adding up the rows of T corresponding to the vertices of V, one therefore obtains the same vector which proves that the rows of T are linearly dependent, implying that its determinant is zero. This proves the total unimodularity of A. $\triangle$

We conclude with a technical remark. One should first remove one of the rows of A before applying Theorem 3.12 since, as such, it does not have full row rank and this fact was implicitly used in the definition of a bfs. However, deleting a row of A still preserves its totally unimodularity.

Exercise 3-15. If A is totally unimodular then A^T is totally unimodular.

Exercise 3-16. Use total unimodularity to prove König's theorem.

The following theorem gives a necessary and sufficient condition for a matrix to be totally unimodular.

Theorem 3.14 *Let A be a $m \times n$ matrix with entries in $\{-1, 0, 1\}$. Then A is TU if and only if for all subsets $R \subseteq \{1, 2, \cdots, n\}$ of rows, there exists a partition of R into R_1 and R_2 such that for all $j \in \{1, 2, \cdots, m\}$:*

$$\sum_{i \in R_1} a_{ij} - \sum_{i \in R_2} a_{ij} \in \{0, 1, -1\}.$$

We will prove only the *if* direction (but that is the most important as this allows to prove that a matrix is totally unimodular).

Proof: Assume that, for every R, the desired partition exists. We need to prove that the determinant of any $k \times k$ submatrix of A is in $\{-1, 0, 1\}$, and this must be true for any k. Let us prove it by induction on k. It is trivially true for $k = 1$. Assume it is true for $k - 1$ and we will prove it for k.

Let B be a $k \times k$ submatrix of A, and we can assume that B is invertible (otherwise the determinant is 0 and there is nothing to prove). The inverse B^{-1} can be written as $\frac{1}{\det(B)} B^*$, where all entries of B^* correspond to $(k-1) \times (k-1)$ submatrices of A. By our inductive hypothesis, all entries of B^* are in $\{-1, 0, 1\}$. Let b_1^* be the first row of B and e_1 be the k-dimensional row vector $[1\ 0\ 0 \cdots 0]$, thus $b_1^* = e_1 B^*$. By the relationship between B and B^*, we have that

$$b_1^* B = e_1 B^* B = \det(B) e_1 B^{-1} B = \det(B) e_1. \tag{5}$$

Let $R = \{i : b_{1i}^* \in \{-1, 1\}\}$. By assumption, we know that there exists a partition of R into R_1 and R_2 such that for all j:

$$\sum_{i \in R_1} b_{ij} - \sum_{i \in R_2} b_{ij} \in \{-1, 0, 1\}. \tag{6}$$

From (5), we have that

$$\sum_{i \in R} b_{1i}^* b_{ij} = \begin{cases} \det(B) & j = 1 \\ 0 & j \neq 1 \end{cases} \tag{7}$$

Since $\sum_{i \in R_1} b_{ij} - \sum_{i \in R_2} b_{ij}$ and $\sum_{i \in R} b_{1i}^* b_{ij}$ differ by a multiple of 2 for each j (since $b_{1i}^* \in \{-1, 1\}$), this implies that

$$\sum_{i \in R_1} b_{ij} - \sum_{i \in R_2} b_{ij} = 0 \quad j \neq 1. \tag{8}$$

For $j = 1$, we cannot get 0 since otherwise B would be singular (we would get exactly the 0 vector by adding and subtracting rows of B). Thus,

$$\sum_{i \in R_1} b_{i1} - \sum_{i \in R_2} b_{i1} \in \{-1, 1\}.$$

If we define $y \in \mathbb{R}^k$ by

$$y_i = \begin{cases} 1 & i \in R_1 \\ -1 & i \in R_2 \\ 0 & otherwise \end{cases}$$

we get that $yB = \pm e_1$. Thus

$$y = \pm e_1 B^{-1} = \pm \frac{1}{\det B} e_1 B^* = \pm \frac{1}{\det B} b_1^*,$$

which implies that $\det B$ must be either 1 or -1. $\triangle$

Exercise 3-17. Suppose we have n activities to choose from. Activity i starts at time t_i and ends at time u_i (or more precisely just before u_i); if chosen, activity i gives us a profit of p_i units. Our goal is to choose a subset of the activities which do not overlap (nevertheless, we can choose an activity that ends at t and one that starts at the same time t) and such that the total profit (i.e. sum of profits) of the selected activities is maximum.

1. Defining x_i as a variable that represents whether activity i is selected ($x_i = 1$) or not ($x_i = 0$), write an integer program of the form $\max\{p^T x : Ax \leq b, x \in \{0,1\}^n\}$ that would solve this problem.
2. Show that the matrix A is totally unimodular, implying that one can solve this problem by solving the linear program $\max\{p^T x : Ax \leq b, 0 \leq x_i \leq 1 \text{ for every } i\}$.

Exercise 3-18. Given a bipartite graph G and given an integer k, let S_k be the set of all incidence vectors of matchings with at most k edges. We are interested in finding a description of $P_k = \text{conv}(S_k)$ as a system of linear inequalities. More precisely, you'll show that $\text{conv}(S_k)$ is given by:

$$\begin{aligned} P_k = \{x : \ & \textstyle\sum_j x_{ij} \leq 1 && \forall i \in A \\ & \textstyle\sum_i x_{ij} \leq 1 && \forall j \in B \\ & \textstyle\sum_i \sum_j x_{ij} \leq k \\ & x_{ij} \geq 0 && i \in A, j \in B\}. \end{aligned}$$

Without the last constraint, we have shown that the resulting matrix is totally unimodular.

1. With the additional constraint, is the resulting matrix totally unimodular? Either prove it or disprove it.
2. Show that P_k indeed equals $\text{conv}(S_k)$.
3. Suppose now that instead of a cardinality constraint on all the edges, our edges are partitioned into E_1 and E_2 and we only impose that our matching has at most k edges from E_1 (and as many as we'd like from E_2). Is it still true that the convex hull of all such matchings is given by simply replacing $\sum_i \sum_j x_{ij} \leq k$ by

$$\sum_i \sum_{j:(i,j)\in E_1} x_{ij} \leq k?$$

4. Lecture notes on flows and cuts

4.1 Maximum Flows

Network flows deals with modelling the flow of a commodity (water, electricity, packets, gas, cars, trains, money, or any abstract object) in a network. The links in the network are capacitated and the commodity does not vanish in the network except at specified locations where we can either inject or extract some amount of commodity. The main question is how much can be sent in this network.

Here is a more formal definition of the maximum flow problem. We have a digraph (directed graph) $G = (V, E)$ and two special vertices s and t; s is called the source and t the sink. We have an upper capacity function $u : E \to \mathbb{R}$ and also a lower capacity function $l : E \to \mathbb{R}$ (sometimes chosen to be 0 everywhere). A flow x will be an assignment of values to the arcs (directed edges) so that:

1. for every $e \in E$: $l(e) \leq x_e \leq u(e)$,
2. for every $v \in V \setminus \{s, t\}$:
$$\sum_{e \in \delta^+(u)} x_e - \sum_{e \in \delta^-(u)} x_e = 0. \tag{1}$$

The notation $\delta^+(u)$ represents the set of arcs *leaving* u, while $\delta^-(u)$ represents the set of arcs *entering* u.

Equations (1) are called *flow conservation* constraints. Given a flow x, its *flow value* $|x|$ is the net flow out of s:
$$|x| := \sum_{e \in \delta^+(s)} x_e - \sum_{e \in \delta^-(s)} x_e. \tag{2}$$

One important observation is that $|x|$ is also equal to the net flow into t, or minus the net flow out of t. Indeed, summing (1) over $u \in V \setminus \{s, t\}$ together with (2), we get:

$$\begin{aligned} |x| &= \sum_{v \in V \setminus \{t\}} \left(\sum_{e \in \delta^+(u)} x_e - \sum_{e \in \delta^-(u)} x_e \right) \\ &= \sum_{e \in \delta^-(t)} x_e - \sum_{e \in \delta^+(t)} x_e \end{aligned}$$

by looking at the contribution of every arc in the first summation.

The *maximum flow problem* is the problem of finding a flow x of maximum value $|x|$. This is a linear program:

$$\begin{aligned} \text{Max} \quad & \sum_{e\in\delta^+(s)} x_e - \sum_{e\in\delta^-(s)} x_e \\ \text{subject to:} \quad & \\ & \sum_{e\in\delta^+(u)} x_e - \sum_{e\in\delta^-(u)} x_e = 0 && u \in V\setminus\{s,t\} \\ & l(e) \le x_e \le u(e) && e \in E. \end{aligned}$$

We could therefore use algorithms for linear programming to find the maximum flow and duality to derive optimality properties, but we will show that more combinatorial algorithms can be developed and duality translates into statements about *cuts*.

In matrix form, the linear program can be written as:

$$\max\{c^T x : \quad \begin{array}{rcl} Nx & = & 0, \\ Ix & \le & u, \\ -Ix & \le & -l\} \end{array}$$

where N is the (vertex-arc incidence[1]) matrix with rows indexed by $u \in V\setminus\{s,t\}$ and columns indexed by arcs $e=(i,j)\in E$; the entry N_{ue} is:

$$N_{ue} = \begin{cases} 1 & u = i \\ -1 & u = j \\ 0 & u \notin \{i,j\}. \end{cases}$$

The constraints of the linear program are thus: $Ax \lesseqgtr b$ where

$$A = \begin{pmatrix} N \\ --- \\ I \\ --- \\ -I \end{pmatrix},$$

and some of the constraints are equalities and some are inequalities.

Lemma 4.1 *A is total unimodular.*

Proof: We could use Theorem 3.14 from the polyhedral chapter, but proving it directly is as easy. Consider any square submatrix of A, and we would like to compute its determinant up to its sign. If there is a row with a single $+1$ or a single -1 in it (in particular, a row coming from either the identity submatrix I or $-I$), we can expand the determinant and

[1]More precisely, part of it as we are not considering vertices s and t

compute the determinant (up to its sign) of a smaller submatrix of A. Repeating this, we now have a square submatrix of N. If there is a column with a single $+1$ or a single -1 then we can expand the determinant along this column and get a smaller submatrix. We are thus left either with an empty submatrix in which case the determinant of the original matrix was $+1$ or -1, or with a square submatrix of N with precisely one $+1$ and one -1 in *every* column. The rows of this submatrix are linearly dependent since their sum is the 0 vector. Thus the determinant is 0. This proves total unimodularity. △

As a corollary, this means that if the right-hand-side (i.e. the upper and lower capacities) are integer-valued then there always exists a maximum flow which takes only integer values.

Corollary 4.2 *If $l : E \to \mathbb{Z}$ and $u : E \to \mathbb{Z}$ then there exists a maximum flow x such that $x_e \in \mathbb{Z}$ for all $e \in E$.*

4.1.1 Special cases

Arc-disjoint paths. If $l(e) = 0$ for all $e \in E$ and $u(e) = 1$ for all $e \in E$, any integer flow x will only take values in $\{0, 1\}$. We claim that for an integer flow x, there exist $|x|$ arc-dsjoint (i.e. not having any arcs in common) paths from s to t. Indeed, such paths can be obtained by *flow decomposition*. As long as $|x| > 0$, take an arc out of s with $x_e = 1$. Now follow this arc and whenever we reach a vertex $u \neq t$, by flow conservation we know that there exists an arc leaving u that we haven't traversed yet (this is true even if we reach s again). This process stops when we reach t and we have therefore identified one path from s to t. Removing this path gives us a new flow x' (indeed flow conservation at vertices $\neq s, t$ is maintained) with $|x'| = |x| - 1$. Repeating this process gives us $|x|$ paths from s to t and, by construction, they are arc-disjoint. The paths we get might not be *simple*[2]; one can however make them simple by removing the part of the walk between repeated occurences of the same vertex. Summarizing, if $l(e) = 0$ for all $e \in E$ and $u(e) = 1$ for all $e \in E$, then from a maximum flow of value k, we can extract k arc-disjoint (simple) paths from s to t. Conversely, if the digraph contains k arc-disjoint paths from s to t, it is easy to construct a flow of value k. This means that the maximum flow value from s to t represents the maximum number of arc-disjoint paths between s and t.

Bipartite matchings. One can formulate the maximum matching problem in a bipartite graph as a maximum flow problem. Indeed, consider a bipartite graph $G = (V, E)$ with bipartition $V = A \cup B$. Consider now a directed graph D with vertex set $V \cup \{s, t\}$. In D, there is an arc from s to every vertex of A with $l(e) = 0$ and $u(e) = 1$. There is also an arc from every vertex in B to t with capacities $l(e) = 0$ and $u(e) = 1$. Every edge $(a, b) \in E$ is oriented from $a \in A$ to $b \in B$ and gets a lower capacity of 0 and an upper capacity equal to $+\infty$ (or just 1). One can easily see that from any matching of size k one can construct a flow of value k; similarly to any *integer valued* flow of value k corresponds a matching of size k. Since the capacities are in $\mathbb{Z}$, by Corollary 4.2, this means that a maximum flow in

[2]A simple path is one in which no vertex is repeated.

D has the same value as the maximum size of any matching in G. Observe that the upper capacities for the arcs between A and B do not matter, provided they are ≥ 1.

Orientations. Consider the problem of orienting the edges of an undirected graph $G = (V, E)$ so that the indegree of any vertex v in the resulting digraph is at most $k(v)$. This can be formulated as a maximum flow problem in which we have (i) a vertex for every vertex of G, (ii) a vertex for every edge of G and (iii) 2 additional vertices s and t. Details are left as an exercise.

Exercise 4-1. Suppose you are given an $m \times n$ matrix $A \in \mathbb{R}^{m \times n}$ with row sums $r_1, \cdots, r_m \in \mathbb{Z}$ and column sums $c_1, \cdots, c_n \in \mathbb{Z}$. Some of the entries might not be integral but the row sums and column sums are. Show that there exists a rounded matrix A' with the following properties:

- row sums and column sums of A and A' are identical,
- $a'_{ij} = \lceil a_{ij} \rceil$ or $a'_{ij} = \lfloor a_{ij} \rfloor$ (i.e. a'_{ij} is a_{ij} either rounded up or down.).

By the way, this rounding is useful to the census bureau as they do not want to publish statistics that would give too much information on specific individuals. They want to be able to modify the entries without modifying row and column sums.

4.2 Cuts

In this section, we derive an important duality result for the maximum flow problem, and as usual, this takes the form of a minmax relation.

In a digraph $G = (V, A)$, we define a *cutset* or more simply a *cut* as the set of arcs $\delta^+(S) = \{(u, v) \in A : u \in S, v \in V \setminus S\}$. Observe that our earlier notation $\delta^+(v)$ for $v \in V$ rather than $\delta^+(\{v\})$ is a slight abuse of notation. Similarly, we define $\delta^-(S)$ as $\delta^+(V \setminus S)$, i.e. the arcs entering the vertex set S. We will typically identify a cutset $\delta^+(S)$ with the corresponding vertex set S. We say that a cut $\delta^+(S)$ is *an $s - t$ cut* (where s and t are vertices) if $s \in S$ and $t \notin S$.

For an undirected graph $G = (V, E)$, $\delta^+(S)$ and $\delta^-(S)$ are identical and will be denoted by $\delta(S) = \{(u, v) \in E : |\{u, v\} \cap S| = 1\}$. Observe that now $\delta(S) = \delta(V \setminus S)$.

For a maximum flow instance on a digraph $G = (V, E)$ and upper and lower capacity functions u and l, we define the capacity $C(S)$ of the cut induced by S as

$$C(S) = \sum_{e \in \delta^+(S)} u(e) - \sum_{e \in \delta^-(S)} l(e) = u(\delta^+(S)) - l(\delta^-(S)).$$

By definition of a flow x, we have that

$$C(S) \geq \sum_{e \in \delta^+(S)} x_e - \sum_{e \in \delta^-(S)} x_e.$$

We have shown earlier that the net flow out of s is equal to the net flow into t. Similarly, we can show that for any S with $s \in S$ and $t \notin S$ (i.e. the cut induced is an $s-t$ cut), we have that the flow value $|x|$ equals:

$$|x| = \sum_{e \in \delta^+(S)} x_e - \sum_{e \in \delta^-(S)} x_e.$$

This is shown by summing (1) over $u \in S \setminus \{s\}$ together with (2). Thus, we get that for any S with $s \in S$ and $t \notin S$ and any $s-t$ flow x, we have:

$$|x| \leq C(S).$$

Therefore, maximizing over the $s-t$ flows and minimizing over the $s-t$ cuts, we get

$$\max |x| \leq \min_{S: s \in S, t \notin S} C(S).$$

This is weak duality, but in fact, one always has equality as stated in the following theorem. Of course, we need the assumption that the maximum flow problem is feasible. For example if there is an edge with $l(e) > u(e)$ then no flow exists (we will show later that a necessary and sufficient condition for the existence of a flow is that (i) $l(e) \leq u(e)$ for every $e \in E$ and (ii) for any $S \subset V$ with $|S \cap \{s,t\}| \neq 1$, we have $u(\delta^+(S)) \geq l(\delta^-(S))$).

Theorem 4.3 (max $s-t$ flow-min $s-t$ cut) *For any maximum flow problem for which a feasible flow exists, we have that that the maximum $s-t$ flow value is equal to the minimum capacity of any $s-t$ cut:*

$$\max_{\text{flow } x} |x| = \min_{S: s \in S, t \notin S} C(S).$$

One way to prove this theorem is by using strong duality of linear programming and show that from any optimum dual solution one can derive an $s-t$ cut of that capacity. Another way, and this is the way we pursue, is to develop an algorithm to find a maximum flow and show that when it terminates we have also a cut whose capacity is equal to the flow we have constructed, therefore proving optimality of the flow and equality in the minmax relation.

Here is an algorithm for finding a maximum flow. Let us assume that we are given a feasible flow x (if $u(e) \geq 0$ and $l(e) \leq 0$ for all e, we could start with $x = 0$). Given a flow x, we define a *residual graph* G_x on the same vertex set V. In G_x, we have an arc (i,j) if (i) $(i,j) \in E$ and $x_{ij} < u((i,j))$ or if (ii) $(j,i) \in E$ and $x_{ji} > l((j,i))$. In case (i), we say that (i,j) is a *forward* arc and in case (ii) it is a *backward* arc. If both (i) and (ii) happen, we introduce two arcs (i,j), one forward and one backward; to be precise, G_x is thus a multigraph. Consider now any directed path P from s to t in the residual graph; such a path is called an *augmenting path*. Let P^+ denote the forward arcs in P, and P^- the backward arcs. We can modify the flow x in the following way:

$$x'_e = \begin{cases} x_e + \epsilon & e \in P^+ \\ x_e - \epsilon & e \in P^- \\ x_e & e \notin P \end{cases}$$

This is known as pushing ϵ units of flow along P, or simply augmenting along P. Oberve that flow conservation at any vertex u still holds when pushing flow along a path. This is trivial if u is not on the path, and if u is on the path, the contributions of the two arcs indicent to u on P cancel each other. To make sure the resulting x' is feasible (satisfies the capacity constraints), we choose

$$\epsilon = \min\left(\min_{e\in P^+}(u(e) - x_e), \min_{e\in P^-} x_e - l(e)\right).$$

By construction of the residual graph we have that $\epsilon > 0$. Thus, pushing ϵ units of flow along an augmenting path provides a new flow x' whose value $|x'|$ satisfy $|x'| = |x| + \epsilon$. Thus the flow x was not maximum.

Conversely, assume that the residual graph G_x does not contain any directed path from s to t. Let $S = \{u \in V : \text{there exists a directed path in } G_x \text{ from } s \text{ to } u\}$. By definition, $s \in S$ and $t \notin S$ (otherwise there would be an augmenting path). Also, by definition, there is no arc in G_x from S to $V \setminus S$. This means that, for $e \in E$, if $e \in \delta^+(S)$ then $x_e = u(e)$ and if $e \in \delta^-(S)$ then $x_e = l(e)$. This implies that

$$C(S) = \sum_{e\in\delta^+(S)} u(e) - \sum_{e\in\delta^-(S)} l(e) = u(\delta^+(S)) - l(\delta^-(S)) = \sum_{e\in\delta^+(S)} x_e - \sum_{e\in\delta^-(S)} x_e = |x|.$$

This shows that the flow x is maximum and there exists an $s-t$ cut of the same capacity as $|x|$.

This almost proves Theorem 4.3. Indeed, as long as there exists an augmenting path, we can push flow along it, update the residual graph and continue. Whenever this algorithm stops, *if it stops*, we have a maximum flow and a corresponding minimum cut. But maybe this algorithm never stops; this can actually happen if the capacities might be irrational and the "wrong" augmenting paths are chosen at every iteration. To complete the proof of the max flow min cut theorem, we can simply use the linear programming formulation of the maximum flow problem and this shows that a maximum flow exists (in a linear program, the max is a real maximum (as it is achieved by a vertex) and not just a supremum which may not be attained). Starting from that flow x and constructing its residual graph G_x, we get that there exists a corresponding minimum $s-t$ cut of the same value.

4.2.1 Interpretation of max flow min cut

The max $s-t$ flow min $s-t$ cut theorem together with integrality of the maximum flow allows to derive several combinatorial min-max relations.

Bipartite matchings. Consider for example the maximum bipartite matching problem and its formulation as a maximum flow problem given in section 4.1.1. We said that for the arcs between A and B we had flexibility on how we choose $u(e)$; here, let us assume we have set them to be equal to $+\infty$ (or any sufficiently large integer). Consider any set

$S \subseteq (\{s\} \cup A \cup B)$ with $s \in S$ (and $t \notin S$). For $C(S)$ to be finite there cannot be any edge $(i,j) \in E$ between $i \in A \cap S$ and $j \in B \setminus S$. In other words, $N(A \cap S) \subseteq B \cap S$, i.e. if we set $C = (A \setminus S) \cup (B \cap S)$ we have that C is a vertex cover. What is the capacity $C(S)$ of the corresponding cut? It is precisely $C(S) = |A \setminus S| + |B \cap S|$, the first term corresponding to the arcs from s to $A \setminus S$ and the second term correspodning to the arcs between $B \cap S$ and t. The max $s-t$ flow min $s-t$ cut theorem therefore implies that there exists a vertex cover C whose cardinality equals the size of the maximum matching. We have thus rederived König's theorem. We could also derive Hall's theorem about the existence of a perfect matching.

Arc-disjoint paths. For the problem of teh maximum number of arc-disjoint paths between s and t, the max $s-t$ flow min $s-t$ cut theorem can be interpreted as Menger's theorem:

Theorem 4.4 *In a directed graph $G = (V, A)$, there are k arc-disjoint paths between s and t if and only if for all $S \subseteq V \setminus \{t\}$ with $s \in S$, we have $|\delta^+(S)| \geq k$.*

Exercise 4-2. At some point during baseball season, each of n teams of the American League has already played several games. Suppose team i has won w_i games so far, and $g_{ij} = g_{ji}$ is the number of games that teams i and j have yet to play. No game ends in a tie, so each game gives one point to either team and 0 to the other. You would like to decide if your favorite team, say team n, can still win. In other words, you would like to determine whether there exists an outcome to the games to be played (remember, with no ties) such that team n has at least as many victories as all the other teams (we allow team n to be tied for first place with other teams).

Show that this problem can be solved as a maximum flow problem. Give a necessary and sufficient condition on the g_{ij}'s so that team n can still win.

Exercise 4-3. Consider the following orientation problem. We are given an undirected graph $G = (V, E)$ and integer values $p(v)$ for every vertex $v \in V$. we would like to know if we can orient the edges of G such that the directed graph we obtain has at most $p(v)$ arcs incoming to v (the "indegree requirements"). In other words, for each edge $\{u, v\}$, we have to decide whether to orient it as (u, v) or as (v, u), and we would like at most $p(v)$ arcs oriented towards v.

1. Show that the problem can be formulated as a maximum flow problem. That is, show how to create a maximum flow problem such that, from its solution, you can decide whether or not the graph can be oriented and if so, it also gives the orientation.
2. Consider the case that the graph cannot be oriented and meet the indegree requirements. Prove from the max-flow min-cut theorem that there must exist a set $S \subseteq V$ such that $|E(S)| > \sum_{v \in S} p(v)$, where as usual $E(S)$ denotes the set of edges with both endpoints within S.

4.3 Efficiency of Maximum Flow Algorithm

The proof of the max $s-t$ flow min $s-t$ cut theorem suggests a simple augmenting path algorithm for finding the maximum flow. Start from any feasible flow and keep pushing flow along an augmenting in the residual graph until no such augmenting path exists. The main question we address now is how many iterations does this algorithm need before terminating.

As mentioned earlier, if the capacities are irrational, this algorithm may never terminate. In the case of integral capacities, if we start from an integral flow, it is easy to see that we always maintain an integral flow and we will always be pushing an integral amount of flow. Therefore, the number of iterations is bounded by the maximum difference between the values of two flows, which is at most $\sum_{e\in\delta(s)}(u(e)-l(e))$. This is finite, but not polynomial in the size of the input (which depends only logarithmically on the capacities u and l).

Shortest augmenting path variant. Edmonds and Karp proposed a variant of the augmenting path algorithm which is guaranteed to terminate in a polynomial number of iterations depending only on $n=|V|$ and $m=|E|$. No assumptions on the capacities are made, and the algorithm is even correct and terminates for irrational capacities.

The idea of Edmonds and Karp is to always find in the residual graph a *shortest* augmenting path, i.e. one with the fewer number of arcs. Given a flow x, consider the residual graph G_x. For any vertex v, let $d(v)$ denote the distance (number of arcs) from s to v in G_x. The shortest augmenting path algorithm is to select a path $v_0-v_1-\cdots-v_k$ in the residual graph where $v_0=s$, $v_k=t$ and $d(v_i)=i$.

The analysis of the algorithm proceeds as follows. Let P be a shotest augmenting path from s to t in G_x and let x' be the resulting flow after pushing as much flow as possible along P. Let d' be the distance labels corresponding to $G_{x'}$. Observe that only reverse arcs (i,j) along P (thus satisfying $d(i)=d(j)+1$) may get introduced in $G_{x'}$. Therefore, after augmentation, we have that $d(j)-d(i)\leq 1$ for every arc $(i,j)\in E_{x'}$. Summing these inqualities along the edges of any path P' in $G_{x'}$ from s to $j\in V$, we get that $d(j)\leq d'(j)$ for any $j\in V$. In particular, we have that $d(t)\leq d'(t)$. As distance labels can never become greater than $n-1$, we have that the distance to t can only increase at most $n-1$ times. But $d'(t)$ can also be equal to $d(t)$. In this case though, the fact that an arc of P is saturated means that there is one fewer arc (i,j) with $d(j)=d(i)+1$ in $G_{x'}$ than in G_x. Thus after at most m such iterations, we must have a strict increase in the distance label of t. Summarizing, this means that the number of augmentations is at most $m(n-1)$. The time it takes to build the residual graph and to find an augmenting path in it is at most $O(m)$ time. This means that the total running time of the shortest augmenting path algorithm is at most $O(m^2n)$. This can be further improved but this is not the focus of these notes.

4.4 Minimum cuts

From now on, we assume that we have only upper capacities u and no lower capacities l ($l(e)=0$ for all e). The minimum $s-t$ cut problem that we have solved so far corresponds

to:

$$\min_{S:s\in S,t\notin S} u(\delta^+(S)).$$

If our graph $G = (V, E)$ is undirected and we would like to find the minimum $s - t$ cut, i.e.

$$\min_{S:s\in S,t\notin S} u(\delta(S)),$$

we can simply replace every edge e by two opposite arcs of the same capacity and reduce the problem to finding a minimum $s - t$ cut in a directed graph. As we have just shown, this can be done by a maximum flow computation.

Now, consider the problem of finding the *global* minimum cut in a graph. Let us first consider the directed case. Finding the global mincut (or just the mincut) means finding S minimizing:

$$\min_{S:\emptyset\neq S\neq V} u(\delta^+(S)).$$

This problem can be reduced to $2(n-1)$ maximum flow computations (where $n = |V|$) in the following way. First we can arbitrarily choose a vertex $s \in V$ and s will either be in S or in $V \setminus S$. Thus, for any $t \in V \setminus \{s\}$, we solve two maximum flow problems, one giving us the minimum $s - t$ cut, the other giving us the minimum $t - s$ cut. Taking the minimum over all such cuts, we get the global mincut in a directed graph.

To find the minimum cut problem in an undirected graph, we do not even need to solve two maximum flow problems for each $t \in V \setminus \{s\}$, only one of them is enough. Thus the global minimum cut problem in an undirected graph can be solved by computing $n - 1$ maximum flow problems. The fastest maximum flow algorithms currently take slightly more than $O(mn)$ time (for example, Goldberg and Tarjan's algorithm [1] take $O(mn \log(n^2/m))$ time). Since we need to use it $n - 1$ times, we can find a mincut in $O(mn^2 \log(n^2/m))$ time. However, these $n - 1$ maxflow problem are related, and Hao and Orlin [2] have shown that it is possible to solve *all* of them in $O(mn \log(n^2/m))$ by modifying Goldberg and Tarjan's algorithm. Thus the minimum cut problem can be solved within this time bound.

We will now derive an algorithm for the mincut problem which is not based on network flows, and which has a running time slightly better than Hao and Orlin's. The algorithm is due to Stoer and Wagner [6], and is a simplification of an earlier result of Nagamochi and Ibaraki [5]. We should also point out that there is a randomized algorithm due to Karger and Stein [4] whose running time is $O(n^2 \log^3 n)$, and a subsequent one due to Karger [3] that runs in $O(m \log^3 n)$.

We first need a definition. Define, for any two sets $A, B \subseteq V$ of vertices,

$$u(A : B) := \sum_{i\in A, j\in B} u((i, j)).$$

The algorithm is described below. In words, the algorithm starts with any vertex, and build an ordering of the vertices by always adding to the selected vertices the vertex whose total cost to the previous vertices is maximized; this is called the *maximum adjacency ordering*. The cut induced by the last vertex in this maximum adjacency ordering is considered,

as well as the cuts obtained by recursively applying the procedure to the graph obtained by shrinking the last two vertices. (If there are edges from a vertex v to these last two vertices then we substitute those two edges with only one edge having capacity equal to the sum of the capacities of the two edges.) The claim is that the best cut among the cuts considered is the overall mincut. The formal description is given below.

MINCUT(G)
- ▷ Let v_1 be any vertex of G
- ▷ $n = |V(G)|$
- ▷ $S = \{v_1\}$
- ▷ **for** $i = 2$ **to** n
  - ▷ let v_i the vertex of $V \setminus S$ s.t.
  - ▷ $c(S : \{v\})$ is maximized (over all $v \in V \setminus S$)
  - ▷ $S := S \cup \{v_i\}$
- ▷ **endfor**
- ▷ **if** $n = 2$ **then** return the cut $\delta(\{v_n\})$
- ▷ **else**
  - ▷ Let G' be obtained from G by shrinking v_{n-1} and v_n
  - ▷ Let C be the cut returned by MINCUT(G')
  - ▷ Among C and $\delta(\{v_n\})$ return the smaller cut (in terms of cost)
- ▷ **endif**

The analysis is based on the following crucial claim.

Claim 4.5 *$\{v_n\}$ (or $\{v_1, v_2, ..., v_{n-1}\}$) induces a min (v_{n-1}, v_n)-cut in G. (Notice that we do not know in advance v_{n-1} and v_n.)*

From this, the correctness of the algorithm follows easily. Indeed, the mincut is either a (v_{n-1}, v_n)-cut or not. If it is, we are fine thanks to the above claim. If it is not, we can assume by induction on the size of the vertex set that it will be returned by the call MINCUT(G').

Proof: Let $v_1, v_2, ..., v_i, ..., v_j, ..., v_{n-1}, v_n$ be the sequence of vertices chosen by the algorithm and let us denote by A_i the sequence $v_1, v_2, ..., v_{i-1}$. We are interested in the cuts that separate v_{n-1} and v_n. Let C be any set such that $v_{n-1} \in C$ and $v_n \notin C$. Then we want to prove that the cut induced by C satisfies

$$u(\delta(C)) \geq u(\delta(A_n)).$$

Let us define vertex v_i to be critical with respect to C if either v_i or v_{i-1} belongs to C but not both. We claim that if v_i is critical then

$$u(A_i : \{v_i\}) \leq u(C_i : A_i \cup \{v_i\} \setminus C_i)$$

where $C_i = (A_i \cup \{v_i\}) \cap C$.

Notice that this implies that $u(\delta(C)) \geq u(\delta(A_n))$ because v_n is critical. Now let us prove the claim by induction on the sequence of critical vertices.

Let v_i be the first critical vertex. Then

$$u(A_i : \{v_i\}) = u(C_i : A_i \cup \{v_i\} \setminus C_i)$$

Thus the base of the induction is true.

For the inductive step, let the assertion be true for critical vertex v_i and let v_j be the next (after v_i) critical vertex. Then

$$\begin{aligned} u(A_j : \{v_j\}) &= u(A_i : \{v_j\}) + u(A_j \setminus A_i : \{v_j\}) \\ &\leq u(A_i : \{v_i\}) + u(A_j \setminus A_i : \{v_j\}) \\ &\leq u(C_i : A_i \cup \{v_i\} \setminus C_i) + u(A_j \setminus A_i : \{v_j\}) \\ &\leq u(C_j : A_j \cup \{v_j\} \setminus C_j), \end{aligned}$$

the first inequality following from the definition of v_i, the second inequality from the inductive hypothesis, and the last from the fact that v_j is the next critical vertex. The proof is concluded observing that A_n induces the cut $\{v_1, v_2, \cdots, v_{n-1}\} : \{v_n\}$. △

The running time depends on the particular implementation. Using Fibonacci heaps we can implement each iteration in $O(m + n \log n)$ time and this yields a total running time of $O(mn + n^2 \log n)$.

Exercise 4-4. Let G be an undirected graph in which the degree of every vertex is at least k. Show that there exist two vertices s and t with at least k edge-disjoint paths between them.

4.5 Minimum T-odd cut problem

Given a graph $G = (V, E)$ with nonnegative edge capacities given by u and an even set T of vertices, the minimum T-odd cut problem is to find S minimizing:

$$\min_{S \subset V : |S \cap T| \text{ odd}} u(\delta(S)).$$

We'll say that S is T-odd if $|S \cap T|$ is odd. Observe that if S is T-odd, so is $V \setminus S$ and vice versa.

We give a polynomial-time algorithm for this problem. We won't present the most efficient one, but one of the easiest ones. Let $ALG(G, T)$ denote this algorithm. The first step of $ALG(G, T)$ is to find a minimum cut having at least one vertex of T on each side:

$$\min_{S \subset V : \emptyset \neq S \cap T \neq T} u(\delta(S)).$$

This can be done by doing $|T| - 1$ minimum $s - t$ cut computations, by fixing one vertex s in T and then trying all vertices $t \in T \setminus \{s\}$, and then returning the smallest cut S obtained in this way.

Now, two things can happen. Either S is a T-odd cut in which case it must be minimum and we are done, or S is T-even (i.e. $T \cap S$ has even cardinality). If S is T-even, we show in the lemma below that we can assume that the minimum T-even cut A is either a subset of S or a subset of $V \setminus S$. Thus we can find by recursively solving 2 smaller minimum T-odd cut problems, one in the graph $G_1 = G/S$ obtained by shrinking S into a single vertex and letting $T_1 = T \setminus S$ and the other in the graph $G_2 = G/(V \setminus S)$ obtained by shrinking $V \setminus S$ and letting $T_2 = T \setminus (V \setminus S) = T \cap S$. Thus the algorithm makes two calls, $ALG(G_1, T_1)$ and $ALG(G_2, T_2)$ and returns the smallest (in terms of capacity) T-odd cut returned.

At first glance, it is not obvious that this algorithm is polynomial as every call may generate two recursive calls. However, letting $R(k)$ denote an upper bound on the running time of $ALG(G, T)$ for instances with $|T| = k$ (and say $|V| \leq n$), we can see that

1. $R(2) = A$, where A is the time needed for a minimum $s - t$ cut computation,
2. $R(k) \leq \max_{k_1 \geq 2, k_2 \geq 2, k = k_1 + k_2} \left((k-1)A + R(k_1) + R(k_2) \right)$.

By induction, we can see that $R(k) \leq k^2 A$, as this is true for $k = 2$ and the inductive step is also satisfied:

$$\begin{aligned} R(k) &\leq \max_{k_1 \geq 2, k_2 \geq 2, k = k_1 + k_2} \left((k-1)A + k_1^2 A + k_2^2 A \right) \\ &\leq (k-1)A + 4A + (k-2)^2 A \\ &= (k^2 - 3k + 7)A \\ &\leq k^2 A, \end{aligned}$$

for $k \geq 4$. Thus, this algorithm is polynomial.

We are left with stating and proving the following lemma.

Lemma 4.6 *If S is a minimum cut among those having at least one vertex of T on each side, and $|S \cap T|$ is even then there exists a minimum T-odd cut A with $A \subseteq S$ or $A \subseteq V \setminus S$.*

Proof: Let B be any minimum T-odd cut. Partition T into T_1, T_2, T_3 and T_4 as follows: $T_1 = T \setminus (B \cup S)$, $T_2 = (T \cap S) \setminus B$, $T_3 = T \cap B \cap S$, and $T_4 = (T \cap B) \setminus S$. Since by definition of B and S we have that $T_1 \cup T_2 \neq \emptyset$, $T_2 \cup T_3 \neq \emptyset$, $T_3 \cup T_4 \neq \emptyset$ and $T_4 \cup T_1 \neq \emptyset$, we must have that either T_1 and T_3 are non-empty, or T_2 and T_4 are non-empty. Possibly replacing B by $V \setminus B$, we can assume that T_1 and T_3 are non-empty.

By submodularity of the cut function, we know that

$$\sum_{e \in \delta(S)} u(e) + \sum_{e \in \delta(B)} u(e) \geq \sum_{e \in \delta(S \cup B)} u(e) + \sum_{e \in \delta(S \cap B)} u(e). \tag{3}$$

Since $T_1 \neq \emptyset$ and $T_3 \neq \emptyset$, both $S \cup B$ and $S \cap B$ separate vertices of T. Furthermore, one of them has to be T-even and the other T-odd, as $|(S \cap B) \cap T| + |(S \cup B) \cap T| = |T_2| + 2|T_3| + |T_4| = |S \cap T| + |B \cap T|$ is odd. Thus, one of $S \cup B$ and $S \cap B$ has to have a cutvalue no greater than the one of B while the other has a cut value no greater than the one of S. This means that either $S \cap B$ or $S \cup B$ is a minimum T-odd cut. $\triangle$

References

[1] A.V. Goldberg and R.E. Tarjan, "A new approach to the maximum flow problem", *Journal of the ACM*, **35**, 921–940, 1988.

[2] X. Hao and J.B. Orlin, "A faster algorithm for finding the minimum cut in a graph", *Proc. of the 3rd ACM-SIAM Symposium on Discrete Algorithms*, 165–174, 1992.

[3] D. Karger, "Minimum cuts in near-linear time", *Proc. of the 28th STOC*, 56–63, 1996.

[4] D. Karger and C. Stein, "An $\tilde{O}(n^2)$ algorithm for minimum cuts", *Proc. of the 25th STOC*, 757–765, 1993.

[5] H. Nagamochi and T. Ibaraki, "Computing edge-connectivity in multigraphs and capacitated graphs", *SIAM Journal on Discrete Mathematics*, **5**, 54–66, 1992.

[6] M. Stoer and F. Wagner, "A simple mincut algorithm", *Proc. of ESA94*, Lecture Notes in Computer Science, **855**, 141–147, 1994.

5. Matroid optimization

5.1 Definition of a Matroid

Matroids are combinatorial structures that generalize the notion of linear independence in matrices. There are many equivalent definitions of matroids, we will use one that focus on its *independent sets*. A matroid M is defined on a finite ground set E (or $E(M)$ if we want to emphasize the matroid M) and a collection of subsets of E are said to be *independent*. The family of independent sets is denoted by $\mathcal{I}$ or $\mathcal{I}(M)$, and we typically refer to a matroid M by listing its ground set and its family of independent sets: $M = (E, \mathcal{I})$. For M to be a matroid, $\mathcal{I}$ must satisfy two main axioms:

(I_1) if $X \subseteq Y$ and $Y \in \mathcal{I}$ then $X \in \mathcal{I}$,

(I_2) if $X \in \mathcal{I}$ and $Y \in \mathcal{I}$ and $|Y| > |X|$ then $\exists e \in Y \setminus X : X \cup \{e\} \in \mathcal{I}$.

In words, the second axiom says that if X is independent and there exists a larger independent set Y then X can be extended to a larger independent by adding an element of $Y \setminus X$. Axiom (I_2) implies that every *maximal* (inclusion-wise) independent set is maximum; in other words, all maximal independent sets have the same cardinality. A maximal independent set is called a *base* of the matroid.

Examples.

- One trivial example of a matroid $M = (E, \mathcal{I})$ is a **uniform** matroid in which

  $$\mathcal{I} = \{X \subseteq E : |X| \leq k\},$$

  for a given k. It is usually denoted as $U_{k,n}$ where $|E| = n$. A base is any set of cardinality k (unless $k > |E|$ in which case the only base is $|E|$).

  A **free** matroid is one in which all sets are independent; it is $U_{n,n}$.

- Another is a **partition** matroid in which E is partitioned into (disjoint) sets $E_1, E_2, \cdots, E_l$ and

  $$\mathcal{I} = \{X \subseteq E : |X \cap E_i| \leq k_i \text{ for all } i = 1, \cdots, l\},$$

  for some given parameters $k_1, \cdots, k_l$. As an exercise, let us check that (I_2) is satisfied. If $X, Y \in \mathcal{I}$ and $|Y| > |X|$, there must exist i such that $|Y \cap E_i| > |X \cap E_i|$ and this means that adding any element e in $E_i \cap (Y \setminus X)$ to X will maintain independence.

  Observe that M would *not* be a matroid if the sets E_i were *not* disjoint. For example, if $E_1 = \{1, 2\}$ and $E_2 = \{2, 3\}$ with $k_1 = 1$ and $k_2 = 1$ then both $Y = \{1, 3\}$ and $X = \{2\}$ have at most one element of each E_i, but one can't find an element of Y to add to X.

- **Linear** matroids (or representable matroids) are defined from a matrix A, and this is where the term *matroid* comes from. Let E denote the index set of the columns of A. For a subset X of E, let A_X denote the submatrix of A consisting only of those columns indexed by X. Now, define

$$\mathcal{I} = \{X \subseteq E : \text{rank}(A_X) = |X|\},$$

i.e. a set X is independent if the corresponding columns are linearly independent. A base B corresponds to a linearly independent set of columns of cardinality rank(A).

Observe that (I_1) is trivially satisfied, as if columns are linearly independent, so is a subset of them. (I_2) is less trivial, but corresponds to a fundamental linear algebra property. If A_X has full column rank, its columns span a space of dimension $|X|$, and similarly for Y, and therefore if $|Y| > |X|$, there must exist a column of A_Y that is not in the span of the columns of A_X; adding this column to A_X increases the rank by 1.

A linear matroid can be defined over any field $\mathbb{F}$ (not just the reals); we say that the matroid is **representable over** $\mathbb{F}$. If the field is $\mathbb{F}_2$ (field of 2 elements with operations (mod 2)) then the matroid is said to be **binary**. If the field is $\mathbb{F}_3$ then the matroid is said to be **ternary**.

For example, the binary matroid corresponding to the matrix

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

corresponds to $U_{2,3}$ since the sum of the 3 columns is the 0 vector when taking components modulo 2. If A is viewed over the reals or over $\mathbb{F}_3$ then the matroid is the free matroid on 3 elements.

Not every matroid is linear. Among those that are linear, some can be represented over some fields $\mathbb{F}$ but not all. For example, there are binary matroids which are not ternary and vice versa (for example, $U_{2,4}$ is ternary but not binary). Matroids which can be represented over *any* field are called **regular**. One can show that regular matroids are precisely those linear matroids that can be represented over the reals by a totally unimodular marix. (Because of this connection, a deep result of Seymour provides a polynomial-time algorithm for deciding whether a matrix is TU.)

- Here is an example of something that is not a matroid. Take a graph $G = (V, E)$, and let $\mathcal{I} = \{F \subseteq E : F \text{ is a matching}\}$. This is not a matroid since (I_2) is not necessarily satisfied ((I_1) is satisfied[1], however). Consider, for example, a graph on 4 vertices and let $X = \{(2, 3)\}$ and $Y = \{(1, 2), (3, 4)\}$. Both X and Y are matchings, but one cannot add an edge of Y to X and still have a matching.

[1]When (I_1) alone is satisfied, $(E, \mathcal{I})$ is called an *independence system.*

- There is, however, another matroid associated with matchings in a (general, not necessarily bipartite) graph $G = (V, E)$, but this time the ground set of M corresponds to V. In the **matching matroid**, $\mathcal{I} = \{S \subseteq V : S \text{ is covered by some matching } M\}$. In this definition, the matching does not need to cover precisely S; other vertices can be covered as well.

- A very important class of matroids in combinatorial optimization is the class of **graphic** matroids (also called cycle matroids). Given a graph $G = (V, E)$, we define independent sets to be those subsets of edges which are forests, i.e. do not contain any cycles. This is called the graphic matroid $M = (E, \mathcal{I})$, or $M(G)$.

  (I_1) is clearly satisfied. To check (I_2), first notice that if F is a forest then the number of connected components of the graph (V, F) is given by $\kappa(V, F) = |V| - |F|$. Therefore, if X and Y are 2 forests and $|Y| > |X|$ then $\kappa(V, Y) < \kappa(V, X)$ and therefore there must exist an edge of $Y \setminus X$ which connects two different connected components of X; adding this edge to X results in a larger forest. This shows (I_2).

  If the graph G is connected, any base will correspond to a spanning tree T of the graph. If the original graph is disconnected then a base corresponds to taking a spanning tree in each connected component of G.

  A graphic matroid is a linear matroid. We first show that the field $\mathbb{F}$ can be chosen to be the reals. Consider the matrix A with a row for each vertex $i \in V$ and a column for each edge $e = (i, j) \in E$. In the column corresponding to (i, j), all entries are 0, except for a 1 in i or j (arbitrarily) and a -1 in the other. To show equivalence between the original matroid M and this newly constructed linear matroid M', we need to show that any independent set for M is independent in M' and vice versa. This is left as an exercise.

  In fact, a graphic matroid is *regular*; it can be represented over any field $\mathbb{F}$. In fact the above matrix A can be shown to be TU. To obtain a representation for a field $\mathbb{F}$, one simply needs to take the representation given above for $\mathbb{R}$ and simply view/replace all -1 by the additive inverse of 1.

5.1.1 Circuits

A minimal (inclusionwise) dependent set in a matroid is called a *circuit*. In a graphic matroid $M(G)$, a circuit will be the usual notion of a *cycle* in the graph G; to be dependent in the graphic matroid, one needs to contain a cycle and the minimal sets of edges containing a cycle are the cycles themselves. In a partition matroid, a circuit will be a set $C \subseteq E_i$ for some i with $|C \cap E_i| = k_i + 1$.

By definition of a circuit C, we have that if we remove any element of a circuit then we get an independent set. A crucial property of circuit is given by the following property,

Theorem 5.1 (Unique Circuit Property) *Let $M = (E, \mathcal{I})$ be a matroid. Let $S \in \mathcal{I}$ and e such that[2] $S + e \notin \mathcal{I}$. Then there exists a unique circuit $C \subseteq S + e$.*

The uniqueness is very important. Indeed, if we consider any $f \in C$ where C is this unique circuit then we have that $C + e - f \in \mathcal{I}$. Indeed, if $C + e - f$ was dependent, it would contain a circuit C' which is distinct from C since $f \notin C'$, a contradiction.

As a special case of the theorem, consider a graphic matroid. If we add an edge to a forest and the resulting graph has a cycle then it has a unique cycle.

Proof:

Suppose $S+e$ contains more than one circuit, say C_1 and C_2 with $C_1 \neq C_2$. By minimality of C_1 and C_2, we have that there exists $f \in C_1 \setminus C_2$. Since $C_1 - f \in \mathcal{I}$ (by minimality of the circuit C_1), we can extend it to a maximal independent set X of $S + e$. Since S is also independent, we must have that $|X| = |S|$ and since $e \in C_1 - f$, we must have that $X = S + e - f \in \mathcal{I}$. But this means that $C_2 \subseteq S + e - f = X$ which is a contradiction since C_2 is dependent. △

Exercise 5-1. Show that any partition matroid is also a linear matroid over $\mathbb{F} = \mathbb{R}$. (No need to give a precise matrix A representing it; just argue its existence.)

Exercise 5-2. Prove that a matching matroid is indeed a matroid.

Exercise 5-3. Show that $U_{2,4}$ is representable over $\mathbb{F}_3$.

Exercise 5-4. Consider the linear matroid (over the reals) defined by the 3×5 matrix:

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

The ground set $E = \{1, 2, 3, 4, 5\}$ has cardinality 5, corresponds to the columns of A, and the independent sets are the set of columns which are linearly independent (over the reals).

1. Give all bases of this matroid.
2. Give all circuits of this matroid.
3. Choose a base B and an element e not in B, and verify the unique circuit property for $B + e$.

Exercise 5-5. Given a family $A_1, A_2, \cdots, A_n$ of sets (they are not necessarily disjoint), a *transversal* is a set T such that $T = \{a_1, a_2, \cdots, a_n\}$, the a_i's are distinct, and $a_i \in A_i$ for all i. A partial transversal is a transversal for $A_{i_1}, A_{i_2}, \cdots, A_{i_k}$ for some subfamily of the A_i's.

Show that the family of all partial transversals forms a matroid (on the ground set $E = \cup A_i$). (Hint: Think of bipartite matchings.)

[2] For a set S and an element e, we often write $S + e$ for $S \cup \{e\}$ and $S - e$ for $S \setminus \{e\}$.

Exercise 5-6. Let $M = (E, \mathcal{I})$ be a matroid. Let $k \in \mathbb{N}$ and define

$$\mathcal{I}_k = \{X \in \mathcal{I} : |X| \leq k\}.$$

Show that $M_k = (E, \mathcal{I}_k)$ is also a matroid. This is known as a truncated matroid.

Exercise 5-7. A family $\mathcal{F}$ of sets is said to be *laminar* if, for any two sets $A, B \in \mathcal{F}$, we have that either (i) $A \subseteq B$, or (ii) $B \subseteq A$ or (iii) $A \cap B = \emptyset$. Suppose that we have a laminar family $\mathcal{F}$ of subsets of E and an integer $k(A)$ for every set $A \in \mathcal{F}$. Show that $(E, \mathcal{I})$ defines a matroid (a *laminar* matroid) where:

$$\mathcal{I} = \{X \subseteq E : |X \cap A| \leq k(A) \text{ for all } A \in \mathcal{F}\}.$$

5.2 Matroid Optimization

Given a matroid $M = (E, \mathcal{I})$ and a cost function $c : E \to \mathbb{R}$, we are interested in finding an independent set S of M of maximum total cost $c(S) = \sum_{e \in S} c(e)$. This is a fundamental problem.

If all $c(e) \geq 0$, the problem is equivalent to finding a maximum cost *base* in the matroid. If $c(e) < 0$ for some element e then, because of (I_1), e will not be contained in any optimum solution, and thus we could eliminate such an element from the ground set. In the special case of a graphic matroid $M(G)$ defined on a connected graph G, the problem is thus equivalent to the maximum spanning tree problem which can be solved by a simple greedy algorithm. This is actually the case for any matroid and this is the topic of this section.

The greedy algorithm we describe actually returns, for every k, a set S_k which maximizes $c(S)$ over all independent sets of size k. The overall optimum can thus simply be obtained by outputting the best of these. The greedy algorithm is the following:

- Sort the elements (and renumber them) such that $c(e_1) \geq c(e_2) \geq \cdots \geq c(e_{|E|})$
- $S_0 = \emptyset$, k=0
- For $j = 1$ to $|E|$
    - if $S_k + e_j \in \mathcal{I}$ then
        - $k \leftarrow k + 1$
        - $S_k \leftarrow S_{k-1} + e_j$
        - $s_k \leftarrow e_j$
- Output $S_1, S_2, \cdots, S_k$

Theorem 5.2 *For any matroid $M = (E, \mathcal{I})$, the greedy algorithm above finds, for every k, an independent set S_k of maximum cost among all independent sets of size k.*

Proof: Suppose not. Let $S_k = \{s_1, s_2, \cdots, s_k\}$ with $c(s_1) \geq c(s_2) \geq \cdots \geq c(s_k)$, and suppose T_k has greater cost ($c(T_k) > c(S_k)$) where $T_k = \{t_1, t_2, \cdots, t_k\}$ with $c(t_1) \geq c(t_2) \geq \cdots \geq c(t_k)$. Let p be the first index such that $c(t_p) > c(s_p)$. Let $A = \{t_1, t_2, \cdots, t_p\}$ and $B = \{s_1, s_2, \cdots, s_{p-1}\}$. Since $|A| > |B|$, there exists $t_i \notin B$ such that $B + t_i \in \mathcal{I}$. Since

$c(t_i) \geq c(t_p) > c(s_p)$, t_i should have been selected when it was considered. To be more precise and detailed, when t_i was considered, the greedy algorithm checked whether t_i could be added to the current set at the time, say S. But since $S \subseteq B$, adding t_i to S should have resulted in an independent set (by (I_1)) since its addition to B results in an independent set. This gives the contradiction and completes the proof. △

Observe that, as long as $c(s_k) \geq 0$, we have that $c(S_k) \geq c(S_{k-1})$. Therefore, to find a maximum cost set over all independent sets, we can simply replace the loop

▷ For $j = 1$ to $|E|$

by

▷ For $j = 1$ to q

where q is such that $c(e_q) \geq 0 > c(e_{q+1})$, and output the last S_k.

For the maximum cost spanning tree problem, the greedy algorithm reduces to Kruskal's algorithm which considers the edges in non-increasing cost and add an edge to the previously selected edges if it does not form a cycle.

One can show that the greedy algorithm actually characterizes matroids. If M is an independence system, i.e. it satisfies (I_1), then M is a matroid if and only if the greedy algorithm finds a maximum cost set of size k for every k and every cost function.

Exercise 5-8. We are given n jobs that each take one unit of processing time. All jobs are available at time 0, and job j has a profit of c_j and a deadline d_j. The profit for job j will only be earned if the job completes by time d_j. The problem is to find an ordering of the jobs that maximizes the total profit. First, prove that if a subset of the jobs can be completed on time, then they can also be completed on time if they are scheduled in the order of their deadlines. Now, let $E(M) = \{1, 2, \cdots, n\}$ and let $\mathcal{I}(M) = \{J \subseteq E(M) : J$ can be completed on time $\}$. Prove that M is a matroid and describe how to find an optimal ordering for the jobs.

5.3 Rank Function of a Matroid

Similarly to the notion of rank for matrices, one can define a rank function for any matroid. The rank function of M, denoted by either $r(\cdot)$ or $r_M(\cdot)$, is defined by:

$$r_M : 2^E \to \mathbb{N} : r_M(X) = \max\{|Y| : Y \subseteq X, Y \in \mathcal{I}\}.$$

Here are a few specific rank functions:

- For a linear matroid, the rank of X is precisely the rank in the linear algebra sense of the matrix A_X corresponding to the columns of A in X.
- For a partition matroid $M = (E, \mathcal{I})$ where

$$\mathcal{I} = \{X \subseteq E : |X \cap E_i| \leq k_i \text{ for } i = 1, \cdots, l\}$$

(the E_i's forming a partition of E) its rank function is given by:

$$r(X) = \sum_{i=1}^{l} \min(|E_i \cap X|, k_i).$$

- For a graphic matroid $M(G)$ defined on graph $G = (V, E)$, the rank function is equal to:

  $$r_{M(G)}(F) = n - \kappa(V, F),$$

  where $n = |V|$ and $\kappa(V, F)$ denotes the number of connected components (including isolated vertices) of the graph with edges F.

The rank function of any matroid $M = (E, \mathcal{I})$ has the following properties:

(R_1) $0 \leq r(X) \leq |X|$ and is integer valued for all $X \subseteq E$

(R_2) $X \subseteq Y \Rightarrow r(X) \leq r(Y)$,

(R_3) $r(X) + r(Y) \geq r(X \cap Y) + r(X \cup Y)$.

The last property is called *submodularity* and is a key concept in combinatorial optimization. It is clear that, as defined, any rank function satisfies (R_1) and (R_2). Showing that the rank function satisfies submodularity needs a proof.

Lemma 5.3 *The rank function of any matroid is submodular.*

Proof: Consider any two sets $X, Y \subseteq E$. Let J be a maximal independent subset of $X \cap Y$; thus, $|J| = r(X \cap Y)$. By (I_2), J can be extended to a maximal (thus maximum) independent subset of X, call it J_X. We have that $J \subseteq J_X \subseteq X$ and $|J_X| = r(X)$. Furthermore, by maximality of J within $X \cap Y$, we know

$$J_X \setminus Y = J_X \setminus J. \tag{1}$$

Now extend J_X to a maximal independent set J_{XY} of $X \cup Y$. Thus, $|J_{XY}| = r(X \cup Y)$.

In order to be able to prove that

$$r(X) + r(Y) \geq r(X \cap Y) + r(X \cup Y)$$

or equivalently

$$|J_X| + r(Y) \geq |J| + |J_{XY}|,$$

we need to show that $r(Y) \geq |J| + |J_{XY}| - |J_X|$. Observe that $J_{XY} \cap Y$ is independent (by (I_1)) and a subset of Y, and thus $r(Y) \geq |J_{XY} \cap Y|$. Observe now that

$$J_{XY} \cap Y = J_{XY} \setminus (J_X \setminus Y) = J_{XY} \setminus (J_X \setminus J),$$

the first equality following from the fact that J_X is a maximal independent subset of X and the second equality by (1). Therefore,

$$r(Y) \geq |J_{XY} \cap Y| = |J_{XY} \setminus (J_X \setminus J)| = |J_{XY}| - |J_X| + |J|,$$

proving the lemma. $\triangle$

5.3.1 Span

The following definition is also motivated by the linear algebra setting.

Definition 5.1 *Given a matroid $M = (E, \mathcal{I})$ and given $S \subseteq E$, let*

$$\text{span}(S) = \{e \in E : r(S \cup \{e\}) = r(S)\}.$$

The span of a set is also called the closure. Observe that $S \subseteq \text{span}(S)$. We claim that $r(S) = r(\text{span}(S))$; in other words, if adding an element to S does not increase the rank, adding many such elements also does not increase the rank. Indeed, take a maximal independent subset of S, say J. If $r(\text{span}(S)) > |J|$ then there exists $e \in \text{span}(S) \setminus J$ such that $J + e \in \mathcal{I}$. Thus $r(S + e) \geq r(J + e) = |J| + 1 > |J| = r(S)$ contradicting the fact that $e \in \text{span}(S)$.

Definition 5.2 *A set S is said to be closed if $S = \text{span}(S)$.*

Exercise 5-9. Given a matroid M with rank function r and given an integer $k \in \mathbb{N}$, what is the rank function of the truncated matroid M_k (see Exercise 5-6 for a definition).

Exercise 5-10. What is the rank function of a laminar matroid, see exercise 5-7?

5.4 Matroid Polytope

Let

$$X = \{\chi(S) \in \{0,1\}^{|E|} : S \in \mathcal{I}\}$$

denote the incidence (or characteristic) vectors of all independent sets of a matroid $M = (E, \mathcal{I})$, and let the *matroid polytope* be defined as $\text{conv}(X)$. In this section, we provide a complete characterization of $\text{conv}(X)$ in terms of linear inequalities. In addition, we illustrate the different techniques proposed in the polyhedral chapter for proving a complete description of a polytope.

Theorem 5.4 *Let*

$$P = \{x \in \mathbb{R}^{|E|} : \begin{array}{ll} x(S) \leq r(S) & \forall S \subseteq E \\ x_e \geq 0 & \forall e \in E\} \end{array}$$

where $x(S) := \sum_{e \in S} x_e$. Then $\text{conv}(X) = P$.

It is clear that $\text{conv}(X) \subseteq P$ since $X \subseteq P$. The harder part is to show that $P \subseteq \text{conv}(X)$. In the next three subsections, we provide three different proofs based on the three techniques to prove complete polyhedral descriptions.

5.4.1 Algorithmic Proof

Here we provide an algorithmic proof based on the greedy algorithm. From $\text{conv}(X) \subseteq P$, we know that

$$\max\{c^T x : x \in X\} = \max\{c^T x : x \in \text{conv}(X)\} \quad \leq \max\{c^T x : \begin{array}{ll} x(S) \leq r(S) & S \subseteq E \\ x_e \geq 0 & e \in E\}. \end{array}$$

Using LP duality, we get that this last expression equals:

$$\min\{\textstyle\sum_S r(S) y_S : \begin{array}{ll} \sum_{S: e \in S} y_S \geq c(e) & \forall e \in E \\ y_S \geq 0 & S \subseteq E\}. \end{array}$$

Our goal now is, for any cost function c, to get an independent set S and a dual feasible solution y such that $c^T \chi(S) = \sum_S r(S) y_S$ which proves that $\text{conv}(X) = P$.

Consider any cost function c. We know that the maximum cost independent set can be obtained by the greedy algorithm. More precisely, it is the last set S_k returned by the greedy algorithm when we consider only those elements up to e_q where $c(e_q) \geq 0 \geq c(e_{q+1})$. We need now to exhibit a dual solution of the same value as S_k. There are exponentially many variables in the dual, but this is not a problem. In fact, we will set most of them to 0.

For any index $j \leq k$, we have $S_j = \{s_1, s_2, \cdots, s_j\}$, and we define U_j to be all elements in our ordering up to and excluding s_{j+1}, i.e. $U_j = \{e_1, e_2, \cdots, e_l\}$ where $e_{l+1} = s_{j+1}$. In other words, U_j is all the elements in the ordering just before s_{j+1}. One important property of U_j is that

$$r(U_j) = r(S_j) = j.$$

Indeed, by independence $r(S_j) = |S_j| = j$, and by (R_2), $r(U_j) \geq r(S_j)$. If $r(U_j) > r(S_j)$, there would be an element say $e_p \in U_j \setminus S_j$ such that $S_j \cup \{e_p\} \in \mathcal{I}$. But the greedy algorithm would have selected that element (by (I_1)) contradicting the fact that $e_p \in U_j \setminus S_j$.

Set the non-zero entries of y_S in the following way. For $j = 1, \cdots, k$, let

$$y_{U_j} = c(s_j) - c(s_{j+1}),$$

where it is understood that $c(s_{k+1}) = 0$. By the ordering of the $c(\cdot)$, we have that $y_S \geq 0$ for all S. In addition, for any $e \in E$, we have that

$$\sum_{S: e \in S} y_S = \sum_{j=t}^{k} y_{U_j} = c(s_t) \geq c(e),$$

where t is the least index such that $e \in U_t$ (implying that e does not come before s_t in the ordering). This shows that y is a feasible solution to the dual. Moreover, its dual value is:

$$\sum_S r(S) y_S = \sum_{j=1}^{k} r(U_j) y_{U_j} = \sum_{j=1}^{k} j(c(s_j) - c(s_{j+1})) = \sum_{j=1}^{k} (j-(j-1)) c(s_j) = \sum_{j=1}^{k} c(s_j) = c(S_k).$$

This shows that the dual solution has the same value as the independent set output by the greedy algorithm, and this is true for all cost functions. This completes the algorithmic proof.

5.4.2 Vertex Proof

Here we will focus on any vertex x of

$$P = \{x \in \mathbb{R}^{|E|} : \begin{array}{ll} x(S) \leq r(S) & \forall S \subseteq E \\ x_e \geq 0 & \forall e \in E\} \end{array}$$

and show that x is an integral vector. Since $x(\{e\}) \leq r(\{e\}) \leq 1$, we get that $x \in \{0,1\}^{|E|}$ and thus it is the incidence vector of an independent set.

Given any $x \in P$, consider the *tight* sets S, i.e. those sets for which $x(S) = r(S)$. The next lemma shows that these tight sets are closed under taking intersections or unions. This lemma is really central, and follows from submodularity.

Lemma 5.5 *Let $x \in P$. Let*

$$\mathcal{F} = \{S \subseteq E : x(S) = r(S)\}.$$

Then

$$S \in \mathcal{F}, T \in \mathcal{F} \Rightarrow S \cap T \in \mathcal{F}, S \cup T \in \mathcal{F}.$$

Observe that the lemma applies even if S and T are disjoint. In that case, it says that $\emptyset \in \mathcal{F}$ (which is always the case as $x(\emptyset) = 0 = r(\emptyset)$) and $S \cup T \in \mathcal{F}$.

Proof: The fact that $S, T \in \mathcal{F}$ means that:

$$r(S) + r(T) = x(S) + x(T). \tag{2}$$

Since $x(S) = \sum_{e \in S} x_e$, we have that

$$x(S) + x(T) = x(S \cap T) + x(S \cup T), \tag{3}$$

i.e. that the function $x(\cdot)$ is modular (both x and $-x$ are submodular). Since $x \in P$, we know that $x(S \cap T) \leq r(S \cap T)$ (this is true even if $S \cap T = \emptyset$) and similarly $x(S \cup T) \leq r(S \cup T)$; this implies that

$$x(S \cap T) + x(S \cup T) \leq r(S\cap) + r(S \cup T). \tag{4}$$

By submodularity, we have that

$$r(S \cap T) + r(S \cup T) \leq r(S) + r(T). \tag{5}$$

Combining (2)–(5), we get

$$r(S) + r(T) = x(S) + x(T) = x(S \cap T) + x(S \cup T) \leq r(S \cap T) + r(S \cup T) \leq r(S) + r(T),$$

and therefore we have equality throughout. This implies that $x(S \cap T) = r(S \cap T)$ and $x(S \cup T) = r(S \cup T)$, i.e. $S \cap T$ and $S \cup T$ in $\mathcal{F}$. △

To prove that any vertex or extreme point of P is integral, we first characterize any face of P. A *chain* $\mathcal{C}$ is a family of sets such that for all $S, T \in \mathcal{C}$ we have that either $S \subseteq T$ or $T \subseteq S$ (or both if $S = T$).

Theorem 5.6 *Consider any face F of P. Then there exists a chain $\mathcal{C}$ and a subset $J \subseteq E$ such that:*

$$F = \{x \in \mathbb{R}^{|E|} : \begin{array}{ll} x(S) \leq r(S) & \forall S \subseteq E \\ x(C) = r(C) & \forall C \in \mathcal{C} \\ x_e \geq 0 & \forall e \in E \setminus J \\ x_e = 0 & \forall e \in J.\} \end{array}$$

Proof: By Theorem 3.5 of the polyhedral notes, we know that any face is characterized by setting some of the inequalities of P by equalities. In particular, F can be expressed as

$$F = \{x \in \mathbb{R}^{|E|} : \begin{array}{ll} x(S) \leq r(S) & \forall S \subseteq E \\ x(C) = r(C) & \forall C \in \mathcal{F} \\ x_e \geq 0 & \forall e \in E \setminus J \\ x_e = 0 & \forall e \in J.\} \end{array}$$

where $J = \{e : x_e = 0 \text{ for all } x \in F\}$ and $\mathcal{F} = \{S : x(S) = r(S) \text{ for all } x \in F\}$. To prove the theorem, we need to argue that the system of equations:

$$x(C) = r(C) \quad \forall C \in \mathcal{F}$$

can be replaced by an equivalent (sub)system in which $\mathcal{F}$ is replaced by a chain $\mathcal{C}$. To be equivalent, we need that

$$\text{span}(\mathcal{F}) = \text{span}(\mathcal{C})$$

where by $\text{span}(\mathcal{L})$ we mean

$$\text{span}(\mathcal{L}) := \text{span}\{\chi(C) : C \in \mathcal{L}\}.$$

Let $\mathcal{C}$ be a maximal subchain of $\mathcal{F}$, i.e. $\mathcal{C} \subseteq \mathcal{F}$, $\mathcal{C}$ is a chain and for all $S \in \mathcal{F} \setminus \mathcal{C}$, there exists $C \in \mathcal{C}$ such that $S \not\subseteq C$ and $C \not\subseteq S$. We claim that $\text{span}(\mathcal{C}) = \text{span}(\mathcal{F})$.

Suppose not, i.e. $H \neq \text{span}(\mathcal{F})$ where $H := \text{span}(\mathcal{C})$. This means that there exists $S \in \mathcal{F} \setminus \mathcal{C}$ such that $\chi(S) \notin H$ but S cannot be added to $\mathcal{C}$ without destroying the chain structure. In other words, for any such S, the set of 'chain violations'

$$V(S) := \{C \in \mathcal{C} : C \not\subseteq S \text{ and } S \not\subseteq C\}$$

is non-empty. Among all such sets S, choose one for which $|V(S)|$ is as small as possible ($|V(S)|$ cannot be 0 since we are assuming that $V(S) \neq \emptyset$ for all possible S). Now fix some set $C \in V(S)$. By Lemma 5.5, we know that both $C \cap S \in \mathcal{F}$ and $C \cup S \in \mathcal{F}$. Observe that there is a linear dependence between $\chi(C), \chi(S), \chi(C \cup T), \chi(C \cap T)$:

$$\chi(C) + \chi(S) = \chi(C \cup S) + \chi(C \cap S).$$

This means that, since $\chi(C) \in H$ and $\chi(S) \notin H$, we must have that either $\chi(C \cup S) \notin H$ or $\chi(C \cap S) \notin H$ (otherwise $\chi(S)$ would be in H). Say that $\chi(B) \notin H$ where B is either $C \cup S$ or $C \cap S$. This is a contradiction since $|V(B)| < |V(S)|$, contradicting our choice of S. Indeed, one can see that $V(B) \subset V(S)$ and $C \in V(S) \setminus V(B)$. $\triangle$

As a corollary, we can also obtain a similar property for an extreme point, starting from Theorem 3.6.

Corollary 5.7 *Let x be any extreme point of P. Then there exists a chain $\mathcal{C}$ and a subset $J \subseteq E$ such that x is the unique solution to:*

$$\begin{array}{ll} x(C) = r(C) & \forall C \in \mathcal{C} \\ x_e = 0 & \forall e \in J. \end{array}$$

From this corollary, the integrality of every extreme point follows easily. Indeed, if the chain given in the corollary consists of $C_1 \subset C_2 \subset C_p$ the the system reduces to

$$\begin{array}{ll} x(C_i \setminus C_{i-1}) = r(C_i) - r(C_{i-1}) & i = 1, \cdots, p \\ x_e = 0 & \forall e \in J, \end{array}$$

where $C_0 = \emptyset$. For this to have a unique solution, we'd better have $|C_i \setminus C_{i-1} \setminus J| \leq 1$ for all i and the values for the resulting x_e's will be integral. Since $0 \leq x_e \leq r(\{e\}) \leq 1$, we have that x is a $0-1$ vector and thus $x = \chi(S)$ for some set S. As $|S| \leq r(S) \leq |S|$, we have $|S| = r(S)$ and thus $S \in \mathcal{I}$ and therefore x is the incidence vector of an independent set. This completes the proof.

5.4.3 Facet Proof

Our last proof of Theorem 5.4 focuses on the facets of $\text{conv}(X)$.

First we need to argue that we are missing any equalities. Let's focus on the (interesting) case in which any singleton set is independent: $\{e\} \in \mathcal{I}$ for every $e \in E$. In that case $dim(\text{conv}(X)) = |E|$ since we can exhibit $|E| + 1$ affinely independent points in X: the 0 vector and all unit vectors $\chi(\{e\})$ for $e \in E$. Thus we do not need any equalities. See exercise 5-11 if we are not assuming that every singleton set is independent.

Now consider any facet F of $\text{conv}(X)$. This facet is induced by a valid inequality $\alpha^T x \leq \beta$ where $\beta = \max\{\sum_{e \in I} \alpha_e : I \in \mathcal{I}\}$. Let

$$\mathcal{O} = \{I \in \mathcal{I} : \sum_{e \in I} \alpha_e = \beta\},$$

i.e. $\mathcal{O}$ is the set of all independent sets whose incidence vectors belong to the face. We'll show that there exists an inequality in our description of P which is satisfied at equality by the incidence vectors of all sets $I \in \mathcal{O}$.

We consider two cases. If there exists $e \in M$ such that $\alpha_e < 0$ then $I \in \mathcal{O}$ implies that $e \notin I$, implying that our face F is included in the face induced by $x_e \geq 0$ (which is in our description of P).

For the other case, we assume that for all $e \in E$, we have $\alpha_e \geq 0$. We can further assume that $\alpha_{\max} := \max_{e \in E} \alpha_e > 0$ since otherwise F is trivial. Now, define S as

$$S = \{e \in E : \alpha_e = \alpha_{\max}\}.$$

Claim 5.8 *For any $I \in \mathcal{O}$, we have $|I \cap S| = r(S)$.*

This means that the face F is contained in the face induced by the inequality $x(S) \leq r(S)$ and therefore we have in our description of P one inequality inducing each facet of $\text{conv}(X)$. Thus we have a complete description of $\text{conv}(X)$.

To prove the claim, suppose that $|I \cap S| < r(S)$. Thus $I \cap S$ can be extended to an independent set $X \in \mathcal{I}$ where $X \subseteq S$ and $|X| > |I \cap S|$. Let $e \in X \setminus (I \cap S)$; observe that $e \in S$ by our choice of X. Since $\alpha_e > 0$ we have that $I + e \notin \mathcal{I}$, thus there is a circuit $C \subseteq I + e$. By the unique circuit property (see Theorem 5.1), for any $f \in C$ we have $I + e - f \in \mathcal{I}$. But $C \setminus S \neq \emptyset$ since $(I \cap S) + e \in \mathcal{I}$, and thus we can choose $f \in C \setminus S$. The cost of $I + e - f$ satisfies:

$$c(I + e - f) = c(I) + c(e) - c(f) > c(I),$$

contradicting the definition of $\mathcal{O}$.

5.5 Facets?

Now that we have a description of the matroid polytope in terms of linear inequalities, one may wonder which of these (exponentially many) inequalities define facets of $\text{conv}(X)$.

For simplicity, let's assume that $r(\{e\}) = 1$ for all $e \in E$ (e belongs to some independent set). Then, every nonnegativity constraint defines a facet of $P = \text{conv}(X)$. Indeed, the 0 vector and all unit vectors except $\chi(\{e\})$ constitute $|E|$ affinely independent points satisfying $x_e = 0$. This mean that the corresponding face has dimension at least $|E| - 1$ and since the dimension of P itself is $|E|$, the face is a facet.

We now consider the constraint $x(S) \leq r(S)$ for some set $S \subseteq E$. If S is not closed (see Definition 5.2) then $x(S) \leq r(S)$ definitely does not define a facet of $P = \text{conv}(X)$ since it is implied by the constraints $x(\text{span}(S)) \leq r(S)$ and $x_e \geq 0$ for $e \in \text{span}(S) \setminus S$.

Another situation in which $x(S) \leq r(S)$ does not define a facet is if S can be expressed as the disjoint union of $U \neq \emptyset$ and $S \setminus U \neq \emptyset$ and $r(U) + r(S \setminus U) = r(S)$. In this case, the inequality for S is implied by those for U and for $S \setminus U$.

Definition 5.3 *S is said to be inseparable if there is no U with $\emptyset \neq U \subset S$ such that $r(S) = r(U) + r(S \setminus U)$.*

From what we have just argued, a necessary condition for $x(S) \leq r(S)$ to define a facet of $P = \text{conv}(X)$ is that S is *closed and inseparable*. This can be shown to be sufficient as well, although the proof is omitted.

As an example, consider a partition matroid with $M = (E, \mathcal{I})$ where

$$\mathcal{I} = \{X \subseteq E : |X \cap E_i| \leq k_i \text{ for all } i = 1, \cdots, l\},$$

for disjoint E_i's. Assume that $k_i \geq 1$ for all i. The rank function for this matroid is:

$$r(S) = \sum_{i=1}^{l} \min(k_i, |S \cap E_i|).$$

For a set S to be inseparable, there must exist (i) $i \in \{1, \cdots, l\}$ with $S \subseteq E_i$, and (ii) $|S \cap E_i|$ is either ≤ 1 or $> k_i$ for every i. Furthermore, for $S \subseteq E_i$ to be closed, we must have that if $|S \cap E_i| \geq k_i$ then $S \cap E_i = E_i$. Thus the only sets we need for the description of a partition matroid polytope are (i) sets $S = E_i$ for i with $|E_i| > k_i$ and (ii) singleton sets $\{e\}$ for $e \in E$. The partition matroid polytope is thus given by:

$$\begin{aligned} P = \{x \in \mathbb{R}^{|E|} : \quad & x(E_i) \leq k_i \quad i \in \{1, \cdots, l\} : |E_i| > k_i \\ & 0 \leq x_e \leq 1 \quad e \in E\}. \end{aligned}$$

As another example, take M to be the graphic matroid $M(G)$. For a set of edges $F \subseteq E$ to be inseparable, we need that the subgraph (V, F) either is a single edge or has only one non-trivial (i.e. with more than 1 vertex) 2-connected component[3] (this is a maximal set of vertices such that for any two vertices in it, there exists two (internally) vertex-disjoint paths between them). Indeed, if we partition F into the edge sets $F_1, \cdots, F_c$ of the (c non-trivial) 2-connected components and single edges (for those edges not in any 2-connected components), we have that $r(F) = \sum_{i=1}^{c} r(F_i)$ and thus c must be 1 for F to be inseparable. Given a set F of edges, its span (with respect to the graphic matroid) consists of all the edges with both endpoints within any connected component of F; these are the edges whose addition does not increase the size of the largest forest. Thus, for F to be inseparable and closed, either it must be a single edge or there exists a vertex set $S \subseteq V$ such that $F = E(S)$ ($E(S)$ denotes all the edges with both endpoints in S) and $(S, E(S))$ is 2-connected. Thus (the minimal description of) the forest polytope (convex hull of all forests in a graph $G = (V, E)$) is given by:

$$\begin{aligned} P = \{x \in \mathbb{R}^{|E|} : \quad & x(E(S)) \leq |S| - 1 \quad S \subseteq V, E(S) \text{ 2-connected or } |S| = 2 \\ & 0 \leq x_e \leq 1 \qquad\quad e \in E\}. \end{aligned}$$

(As usual, $x(E(S))$ denotes $\sum_{e \in E(S)} x_e$.) We could also include all vertex sets S; this would of course also give the spanning tree polytope, albeit not the minimal description. Observe that this polyhedral description still has a very large number of inequalities, wven if we include only the facet-defining inequalities.

From this, we can also easily derive the *spanning tree polytope* of a graph, namely the convex hull of incidence vectors of all spanning trees in a graph. Indeed, this is a face of the forest polytope obtained by replacing the inequality for $S = V$ ($x(E) \leq |V| - 1$) by an equality:

$$\begin{aligned} P = \{x \in \mathbb{R}^{|E|} : \quad & x(E) = |V| - 1 \\ & x(E(S)) \leq |S| - 1 \quad S \subset V, E(S) \text{ 2-connected or } |S| = 2 \\ & 0 \leq x_e \leq 1 \qquad\quad e \in E\}. \end{aligned}$$

[3]For example, if the graph consists of 3 cycles that are vertex-disjoint except for one common (to all three) vertex v then each of these cycles would form a 2-connected component. In general, there is a forest structure on the set of 2-connected components.

Exercise 5-11. Let $M = (E, \mathcal{I})$ be a matroid and let $S = \{e \in E : \{e\} \in \mathcal{I}\}$. Show that $\dim(\text{conv}(X)) = |S|$ (where X is the set of incidence vectors of independent sets) and show that the description for P has the required number of linearly independent equalities.

Exercise 5-12. Let $M = (E, \mathcal{I})$ be a matroid and let P be the corresponding matroid polytope, i.e. the convex hull of characteristic vectors of independent sets. Show that two independent sets I_1 and I_2 are adjacent on P if and only if either (i) $I_1 \subseteq I_2$ and $|I_1|+1 = |I_2|$, or (ii) $I_2 \subseteq I_1$ and $|I_2| + 1 = |I_1|$, or (iii) $|I_1 \setminus I_2| = |I_2 \setminus I_1| = 1$ and $I_1 \cup I_2 \notin \mathcal{I}$.

6. Lecture notes on matroid intersection

One nice feature about matroids is that a simple greedy algorithm allows to optimize over its independent sets or over its bases. At the same time, this shows the limitation of the use of matroids: for many combinatorial optimization problems, the greedy algorithm does not provide an optimum solution. Yet, as we will show in this chapter, the expressive power of matroids become much greater once we consider the *intersection* of the family of independent sets of *two* matroids.

Consider two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ on the same ground set E, and consider the family of indepedent sets common to both matroids, $\mathcal{I}_1 \cap \mathcal{I}_2$. This is what is commonly referred to as the intersection of two matroids.

In this chapter, after giving some examples of matroid intersection, we show that that finding a largest common independent set to 2 matroids can be done efficiently, and provide a min-max relation for the maximum value. We also consider the weighted setting (generalizing the assignment problem), although we will not give an algorithm in the general case (although one exists); we only restrict to a special case, namely the arborescence problem. We shall hint an algorithm for the general case by characterizing the matroid intersection polytope and thereby giving a min-max relation for it (an NP∩co-NP characterization). Finally, we discuss also *matroid union*; a powerful way to construct matroids from other matroids in which matroid intersection plays a central role. (The term 'matroid union' is misleading as it is not what we could expect after having defined matroid intersection... it does *not* correspond to $\mathcal{I}_1 \cup \mathcal{I}_2$.)

6.1 Examples

6.1.1 Bipartite matchings

Matchings in a bipartite graph $G = (V, E)$ with bipartition (A, B) do not form the independent sets of a matroid. However, they can be viewed as the common independent sets to two matroids; this is the canonical example of matroid intersection.

Let M_A be a partition matroid with ground set E where the partition of E is given by $E = \bigcup\{\delta(v) : v \in A\}$ where $\delta(v)$ denotes the edges incident to v. Notice that this is a partition since all edges have precisely one endpoint in A. We also define $k_v = 1$ for every $v \in A$. Thus, the family of independent sets of M_A is given by

$$\mathcal{I}_A = \{F : |F \cap \delta(v)| \leq 1 \text{ for all } v \in A\}.$$

In other words, a set of edges is independent for M_A if it has at most one edge incident to every vertex of A (and any number of edges incident to every vertex of b). We can similarly define $M_B = (E, \mathcal{I}_B)$ by

$$\mathcal{I}_B = \{F : |F \cap \delta(v)| \leq 1 \text{ for all } v \in B\}.$$

Now observe that any $F \in \mathcal{I}_A \cap \mathcal{I}_B$ corresponds to a matching in G, and vice versa. And the largest common independent set to $\mathcal{I}_A$ and $\mathcal{I}_B$ corresponds to a maximum matching in G.

6.1.2 Arborescences

Given a digraph $D = (V, A)$ and a special root vertex $r \in V$, an r-arborescence (or just arborescence) is a spanning tree (when viewed as an undirected graph) directed away from r. Thus, in a r-arborescence, every vertex is reachable from the root r. As an r-arborescence has no arc incoming to the root, we assume that D has no such arc.

r-arborescences can be viewed as sets simultaneously independent in two matroids. Let G denote the undirected counterpart of D obtained by disregarding the directions of the arcs. Note that if we have both arcs $a_1 = (u, v)$ and $a_2 = (v, u)$ in D then we get two undirected edges also labelled a_1 and a_2 between u and v in G. Define $M_1 = (A, \mathcal{I}_1) = M(G)$ the graphic matroid corresponding to G, and $M_2 = (A, \mathcal{I}_2)$ the partition matroid in which independent sets are those with at most one arc incoming to every vertex $v \neq r$. In other words, we let

$$\mathcal{I}_2 = \{F : |F \cap \delta^-(v)| \leq 1 \text{ for all } v \in V \setminus \{r\}\}$$

where $\delta^-(v)$ denotes the set $\{(u, v) \in A\}$ of arcs incoming to v. Thus, any r-arborescence is independent in both matroids M_1 and M_2. Conversely, any set T independent in both M_1 and M_2 and of cardinality $|V| - 1$ (so that it is a base in both matroids) is an r-arborescence. Indeed, such a T being a spanning tree in G has a unique path between r and any vertex v; this path must be directed from the root r since otherwise we would have either an arc incoming to r or two arcs incoming to the same vertex.

In the minimum cost arborescence problem, we are also given a cost function $c : A \to \mathbb{R}$ and we are interested in finding the minimum cost r-arborescence. This is a directed counterpart to the minimum spanning tree problem but, here, the greedy algorithm does not solve the problem.

6.1.3 Orientations

Given an undirected graph $G = (V, E)$, we consider orientations of all its edges into directed arcs; namely, each (undirected) edge[1] $\{u, v\}$ is either replaced by an arc[2] (u, v) from u to v, or by an arc (v, u) from v to u. Our goal is, given $k : V \to \mathbb{N}$, to decide whether there exists an orientation such that, for every vertex $v \in V$, the indegree of vertex v (the number of arcs entering v) is at most $k(v)$. Clearly, this is not always possible, and this problem can be solved using matroid intersection (or network flows as well).

To attack this problem through matroid intersection, consider the directed graph $D = (V, A)$ in which every edge $e = \{u, v\}$ of E is replaced by two arcs (u, v) and (v, u). With the

[1] Usually, we use (u, v) to denote an (undirected) edge. In this section, however, we use the notation $\{u, v\}$ rather than (u, v) to emphasize that edges are undirected.

[2] We use arcs in the case of directed graphs, and edges for undirected graphs.

arc set A as ground set, we define two partition matroids, M_1 and M_2. To be independent in M_1, one can take at most one of $\{(u,v),(v,u)\}$ for every $(u,v) \in E$, i.e.

$$\mathcal{I}_1 = \{F \subseteq A : |F \cap \{(u,v),(v,u)\}| \leq 1 \text{ for all } (u,v) \in E\}.$$

To be independent in M_2, one can take at most $k(v)$ arcs among $\delta^-(v)$ for every v:

$$\mathcal{I}_2 = \{F \subseteq A : |F \cap \delta^-(v)| \leq k(v) \text{ for all } v \in V\}.$$

Observe that this indeed defines a partition matroid since the sets $\delta^-(v)$ over all v partition A.

Therefore, there exists an orientation satisfying the required indegree restrictions if there exists a common independent set to M_1 and M_2 of cardinality precisely $|E|$ (in which case we select either (u,v) or (v,u) but not both).

6.1.4 Colorful Spanning Trees

Suppose we have an undirected graph $G = (V,E)$ and every edge has a color. This is represented by a partition of E into $E_1 \cup \cdots \cup E_k$ where each E_i represents a set of edges of the same color i. The problem of deciding whether this graph has a spanning tree in which all edges have a different color can be tackled through matroid intersection. Such a spanning tree is called *colorful*.

Colorful spanning trees are bases of the graphic matroid $M_1 = M(G)$ which are also independent in the partition matroid $M_2 = (E, \mathcal{I}_2)$ defined by $\mathcal{I}_2 = \{F : |F \cap E_i| \leq 1 \text{ for all } i\}$.

6.1.5 Union of Two Forests

In Section 6.5, we show that one can decide whether a graph G has two edge-disjoint spanning trees by matroid intersection.

6.2 Largest Common Independent Set

As usual, one issue is to find a common independent set of largest cardinality, another is to prove that indeed it is optimal. This is done through a min-max relation.

Given two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ with rank functions r_1 and r_2 respectively, consider any set $S \in \mathcal{I}_1 \cap \mathcal{I}_2$ and any $U \subseteq E$. Observe that

$$|S| = |S \cap U| + |S \cap (E \setminus U)| \leq r_1(U) + r_2(E \setminus U),$$

since both $S \cap U$ and $S \cap (E \setminus U)$ are independent in M_1 and in M_2 (by property (I_1)); in particular (and this seems weaker), $S \cap U$ is independent for M_1 while $S \cap (E \setminus U)$ is independent for M_2. Now, we can take the maximum over S and the minimum over U and derive:

$$\max_{S \in \mathcal{I}_1 \cap \mathcal{I}_2} |S| \leq \min_{U \subseteq E} \left[r_1(U) + r_2(E \setminus U) \right].$$

Somewhat surprisingly, we will show that we always have equality:

Theorem 6.1 (Matroid Intersection) *For any two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ with rank functions r_1 and r_2 respectively, we have:*

$$\max_{S \in \mathcal{I}_1 \cap \mathcal{I}_2} |S| = \min_{U \subseteq E} [r_1(U) + r_2(E \setminus U)]. \tag{1}$$

Before describing an algorithm for matroid intersection that proves this theorem, we consider what the min-max result says for some special cases. First, observe that we can always restrict our attention to sets U which are closed for matroid M_1. Indeed, if that was not the case, we could replace U by $V = \text{span}_{M_1}(U)$ and we would have that $r_1(V) = r_1(U)$ while $r_2(E \setminus V) \leq r_2(E \setminus U)$. This shows that there always exists a set U attaining the minimum which is closed for M_1. Similarly, we could assume that $E \setminus U$ is closed for M_2 (but both assumptions cannot be made simultaneously).

When specializing the matroid intersection theorem to the graph orientation problem discussed earlier in this chapter, we can derive the following.

Theorem 6.2 *$G = (V, E)$ has an orientation such that the indegree of vertex v is at most $k(v)$ for every $v \in V$ if and only if for all $P \subseteq V$ we have[3]:*

$$|E(P)| \leq \sum_{v \in P} k(v).$$

Similarly, for colorful spanning trees, we obtain:

Theorem 6.3 *Given a graph $G = (V, E)$ with edges of E_i colored i for $i = 1, \cdots, k$, there exists a colorful spanning tree if and only if deleting the edges of any c colors (for any $c \in \mathbb{N}$) produces at most $c + 1$ connected components.*

We now prove Theorem 6.1 by exhibiting an algorithm for finding a maximum cardinality independent set common to two matroids and a corresponding set U for which we have equality in (1). For the algorithm, we will start with $S = \emptyset$ and at each step either augment S or produce a U that gives equality. Our algorithm will rely heavily on a structure called the exchange graph. We first focus on just one matroid.

Definition 6.1 *Given a matroid $M = (E, \mathcal{I})$ and an independent set $S \in \mathcal{I}$, the* exchange graph *$\mathcal{G}_M(S)$ (or just $\mathcal{G}(S)$) is the bipartite graph with bipartition S and $E \setminus S$ with an edge between $y \in S$ and $x \in E \setminus S$ if $S - y + x \in \mathcal{I}$.*

Lemma 6.4 *Let S and T be two independent sets in M with $|S| = |T|$. Then there exists a perfect matching between $S \setminus T$ and $T \setminus S$ in $\mathcal{G}_M(S)$.*

The proof is omitted. The converse to Lemma 6.4 does not hold. We next prove a proposition that is a partial converse to the above lemma.

[3] $E(P)$ denotes the set of edges with both endoints in P.

Proposition 6.5 *Let $S \in \mathcal{I}$ with exchange graph $\mathcal{G}_M(S)$. Let T be a set with $|T| = |S|$ and such that $\mathcal{G}_M(S)$ has a* unique *perfect matching between $S \setminus T$ and $T \setminus S$. Then $T \in \mathcal{I}$.*

Proof: Let N be the unique matching. Orient edges in N from $T \setminus S = \{x_1, \cdots, x_t\}$ to $S \setminus T = \{y_1, \cdots, y_t\}$, and orient the rest from $S \setminus T$ to $T \setminus S$. If we contract the edges of N, observe that the resulting directed graph has no directed cycle since, otherwise, we could find an alternating cycle prior to contraction, and this would contradict the uniqueness of the matching. Hence the vertices of $\mathcal{G}_M(S)$ can be numbered (by a topological ordering) so that (i) the endpoints of the matching are numbered consecutively and (ii) all edges are directed from smaller-numbered vertices to larger-numbered vertices. So, number $S \setminus T$ and $T \setminus S$ such that $N = \{(y_1, x_1), (y_2, x_2), \ldots, (y_t, x_t)\}$ and such that (y_i, x_j) is never an edge for $i < j$.

Now suppose for the sake of contradiction that $T \notin \mathcal{I}$. Then T has a circuit C. Take the smallest i such that $x_i \in C$ (there must exist at least one element of C in $T \setminus S$ since $C \subseteq T$ and S is independent). By construction, (y_i, x) is not an edge for $x \in C - x_i$. This implies that $x \in \text{span}(S - y_i)$ for all $x \in C - x_i$. Hence $C - x_i \subseteq \text{span}(S - y_i)$, so $\text{span}(C - x_i) \subseteq \text{span}(\text{span}(S - y_i)) = \text{span}(S - y_i)$. C is a cycle, so $x_i \in \text{span}(C - x_i)$, and thus $x_i \in \text{span}(S - y_i)$. This is a contradiction, since $(y_i, x_i) \in \mathcal{G}_M(S)$ by assumption. Therefore T must be in $\mathcal{I}$, which proves the proposition. $\triangle$

We are now ready to describe the algorithm for proving the minmax formula. First, we define a new type of exchange graph for the case when we are dealing with two matroids.

Definition 6.2 *For $S \in \mathcal{I}_1 \cap \mathcal{I}_2$, the exchange graph $\mathcal{D}_{M_1,M_2}(S)$ is the directed bipartite graph with bipartition S and $E \setminus S$ such that (y, x) is an arc if $S - y + x \in \mathcal{I}_1$ and (x, y) is an arc if $S - y + x \in \mathcal{I}_2$.*

Also define $X_1 := \{x \notin S \mid S+x \in \mathcal{I}_1\}$, the set of *sources*, and $X_2 := \{x \notin S \mid S+x \in \mathcal{I}_2\}$, the set of *sinks*. Then the algorithm is to find a path (we call it an *augmenting path*) from X_1 to X_2 that does not contain any shortcuts (arcs that point from an earlier vertex on the path to a non-adjacent later vertex on the path). This for example can be obtained by selecting a shortest path from X_1 to X_2. Then replace S with $S \triangle P$, where P is the set of vertices on the path. As a special case, if $X_1 \cap X_2 \neq \emptyset$, then we end up with a path that consists of a singleton vertex and we can just add that element to S. If there is no such path, then set $U := \{z \in S \mid z \text{ can reach some vertex in } X_2 \text{ in } \mathcal{D}_{M_1,M_2}(S)\}$. Alternatively, we could define $E \setminus U$ as the set of vertices which can be reached from a vertex in X_1; this may give a different set.

To prove that this algorithm is correct, we need to show that

1. When we stop, the sets S and U do indeed give equality in the minmax formula (1).
2. At each stage in the algorithm, $S \triangle P \in \mathcal{I}_1 \cap \mathcal{I}_2$.

Proof of 1: First note that $X_2 \subseteq U$ and that $X_1 \cap U = \emptyset$ (as otherwise we could keep running the algorithm to increase the size of S). We claim that $r_1(U) = |S \cap U|$ and

$r_2(S \setminus U) = |S \cap (E \setminus U)|$. Together, these would imply that $|S| = r_1(U) + r_2(E \setminus U)$, which is what we need.

Suppose first that $|S \cap U| \neq r_1(U)$. Since $S \cap U \subseteq U$ and $S \cap U$ is independent, this would imply that $|S \cap U| < r_1(U)$. Then there would have to exist some $x \in U \setminus S$ such that $(S \cap U) + x \in \mathcal{I}_1$. As $S \in \mathcal{I}_1$, we can repeatedly add elements of S to $(S \cap U) + x$ and thereby obtain a set of the form $S + x - y$ for some $y \in S \setminus U$ with $S + x - y \in \mathcal{I}_1$. But then (y, x) is an arc in $\mathcal{D}_{M_1,M_2}(S)$, so $y \in U$ (since $x \in U$). This is a contradiction, so we must have $|S \cap U| = r_1(U)$.

Now suppose that $|S \cap (E \setminus U)| \neq r_2(E \setminus U)$. Then as before we must have $|S \cap (E \setminus U)| < r_2(S \setminus U)$. Thus there exists $x \in (E \setminus U) \setminus S$ such that $(S \cap (E \setminus U)) + x \in \mathcal{I}_2$. So, by the same logic as before, we can find $y \in S \setminus (E \setminus U)$ such that $S - y + x \in \mathcal{I}_2$. But $S \setminus (E \setminus U) = S \cap U$, so we have $y \in S \cap U$ such that $S - y + x \in \mathcal{I}_2$. But then (x, y) is an arc in $\mathcal{D}_{M_1,M_2}(S)$, so $x \in U$ (since $y \in U$). This is a contradiction, so we must have $|S \cap (E \setminus U)| = r_2(E \setminus U)$. $\triangle$

Proof of 2: Recall that we need to show that $S \triangle P \in \mathcal{I}_1 \cap \mathcal{I}_2$ whenever P is a path from X_1 to X_2 with no shortcuts. We first show that $S \triangle P \in \mathcal{I}_1$. We start by defininining a new matroid M_1' from M_1 as $M_1' := (E \cup \{t\}, \{J \mid J \setminus \{t\} \in \mathcal{I}_1\}$. In other words, we simply add a new element $\{t\}$ that is independent from all the other elements of the matroid. Then we know that $S \cup \{t\}$ is independent in M_1' and M_2' (where we define M_2' analogously to M_1'). On the other hand, if we view $\mathcal{D}_{M_1'}(S \cup \{t\})$ as a subgraph of $\mathcal{D}_{M_1',M_2'}(S \cup \{t\})$, then there exists a perfect matching in $\mathcal{D}_{M_1'}(S \cup \{t\})$ between $(S \cap P) \cup \{t\}$ and $P \setminus S$ (given by the arcs in P that are also arcs in $\mathcal{D}_{M_1'}(S \cup \{t\})$, together with the arc between $\{t\}$ and the first vertex in P). Furthermore, this matching is unique since P has no shortcuts, so by the proposition we know that $(S \cup \{t\}) \triangle P$ is independent in M_1', hence $S \triangle P$ is independent in M_1.

The proof that $S \triangle P \in \mathcal{I}_2$ is identical, except that this time the matching consists of the arcs in P that are also arcs in $\mathcal{D}_{M_2'}(S \cup \{t\})$, together with the arc between $\{t\}$ and the *last* vertex in P (rather than the first). $\triangle$

So, we have proved that our algorithm is correct, and as a consequence have established the minmax formula.

Exercise 6-1. Deduce König's theorem about the maximum size of a matching in a bipartite graph from the min-max relation for the maximum independent set common to two matroids.

6.3 Matroid Intersection Polytope

In this section, we characterize the *matroid intersection polytope* in terms of linear inequalities, that is the convex hull of characteristic vectors of independent sets common to two matroids. Let $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ be two matroids, and let

$$X = \{\chi(S) \in \{0,1\}^{|E|} : S \in \mathcal{I}_1 \cap \mathcal{I}_2\}.$$

The main result is that $conv(X)$ is precisely given by the intersection of the matroid polytopes for M_1 and M_2.

Theorem 6.6 *Let*

$$P = \{x \in \mathbb{R}^{|E|} : \begin{array}{ll} x(S) \leq r_1(S) & \forall S \subseteq E \\ x(S) \leq r_2(S) & \forall S \subseteq E \\ x_e \geq 0 & \forall e \in E\} \end{array}.$$

Then $conv(X) = P$.

Our proof will be vertex-based. We will show that any extreme point of P is integral, and it can then be easily seen that it corresponds to a common independent set. The proof will rely on total unimodularity in a subtle way. Even though the overall matrix defining P is *not* totally unimodular, we will show that, for every extreme point x^*, x^* can be seen as the solution of a system of equations whose underlying matrix is totally unimodular. This is a powerful approach that can apply to many settings.

Proof: Let x^* be an extreme point of P. We know that x^* is uniquely characterized once we know the inequalities that are tight in the description of P. Let

$$\mathcal{F}_i = \{S \subseteq E : x^*(S) = r_i(S)\},$$

for $i = 1, 2$. Let $E_0 = \{e \in E : x^*_e = 0\}$. We know that x^* is the unique solution to

$$\begin{array}{ll} x(S) = r_1(S) & S \in \mathcal{F}_1 \\ x(S) = r_2(S) & S \in \mathcal{F}_2 \\ x_e = 0 & e \in E_0. \end{array}$$

Consider the matroid polytope P_i for matroid M_i for $i = 1, 2$, and define the face F_i of P_i (for $i = 1, 2$) to be

$$F_i = \{x \in P_i : \begin{array}{ll} x(S) = r_1(S) & \forall S \in \mathcal{F}_i \\ x_e = 0 & \forall e \in E_0\} \end{array}.$$

Observe that $F_1 \cap F_2 = \{x^*\}$. Also, by Theorem 4.6 of the chapter on matroid optimization, we have that F_i can be alternatively defined by a *chain* $\mathcal{C}_i$. Thus, x^* is the unique solution to

$$\begin{array}{ll} x(S) = r_1(S) & S \in \mathcal{C}_1 \\ x(S) = r_2(S) & S \in \mathcal{C}_2 \\ x_e = 0 & e \in E_0. \end{array}$$

After eliminating all variables in E_0, this system can be written as $Ax = b$, where the rows of A are the characteristic vectors of $\mathcal{C}_1 \cup \mathcal{C}_2$.

Such a matrix A is totally unimodular and this can be shown by using Theorem 3.14. Consider any subset of rows; this corresponds to restricting our attention to chains $\mathcal{C}'_1$ and $\mathcal{C}'_2$. Consider first $\mathcal{C}'_1$. If we assign the largest set to R_1 and then keep alternating the assignment between R_2 and R_1 as we consider smaller and smaller sets, we obtain that

$$\sum_{i \in \mathcal{C}'_1 \cap R_1} a_{ij} - \sum_{i \in \mathcal{C}'_1 \cap R_2} a_{ij} \in \{0, 1\},$$

for all j. If for $\mathcal{C}_2'$ we start with the largest set being in R_2, we get

$$\sum_{i \in \mathcal{C}_2' \cap R_1} a_{ij} - \sum_{i \in \mathcal{C}_2' \cap R_2} a_{ij} \in \{0, -1\},$$

for all j. Combining both, we get that indeed for every j, we get a value in $\{0, 1, -1\}$ showing that the matrix is totally unimodular. As a result, x^* is integral, and therefore corresponds to the characteristic vector of a common independent set. $\triangle$

6.4 Arborescence Problem

The minimum cost r-arborescence is the problem of, given a directed graph $D = (V, A)$, a root vertex $r \in V$ and a cost c_a for every arc $a \in A$, finding an r-arborescence in D of minimum total cost. This can thus be viewed as a weighted matroid intersection problem and we could use the full machinery of matroid intersection algorithms and results. However, here, we are going to develop a simpler algorithm using notions similar to the Hungarian method for the assignment problem. We will assume that the costs are nonnegative.

As an integer program, the problem can be formulated as follows. Letting x_a be 1 for the arcs of an r-arborescence, we have the formulation:

$$\begin{aligned} OPT \quad = \quad & \min \sum_{a \in A} c_a x_a \\ & \text{subject to:} \\ & \sum_{a \in \delta^-(S)} x_a \geq 1 && \forall S \subseteq V \setminus \{r\} \\ & \sum_{a \in \delta^-(v)} x_a = 1 && \forall v \in V \setminus \{r\} \\ & x_a \in \{0, 1\} && a \in A. \end{aligned}$$

In this formulation $\delta^-(S)$ represents the set of arcs $\{(u, v) \in A : u \notin S, v \in S\}$. One can check that any feasible solution to the above corresponds to the incidence vector of an r-arborescence. Notice that this optimization problem has an exponential number of constraints. We are going to show that we can relax both the integrality restrictions to $x_a \geq 0$ and also remove the equality constraints $\sum_{a \in \delta^-(v)} x_a = 1$ and still there will be an r-arboresence that will be optimum for this relaxed (now linear) program. The relaxed linear program (still with an exponential number of constraints) is:

$$LP = \min \sum_{a \in A} c_a x_a$$

subject to:

$$(P) \qquad \sum_{a \in \delta^-(S)} x_a \geq 1 \qquad \forall S \subseteq V \setminus \{r\}$$
$$x_a \geq 0 \qquad a \in A.$$

The dual of this linear program is:

$$LP = \max \sum_{S \subseteq V \setminus \{r\}} y_S$$

subject to:

$$(D) \qquad \sum_{S : a \in \delta^-(S)} y_S \leq c_a$$
$$y_S \geq 0 \qquad S \subseteq V \setminus \{r\}.$$

The algorithm will be constructing an arborescence T (and the corresponding incidence vector x with $x_a = 1$ whenever $a \in T$ and 0 otherwise) and a feasible dual solution y which satisfy complementary slackness, and this will show that T corresponds to an optimum solution of (P), and hence is an optimum arborescence. Complementary slackness says:

1. $y_S > 0 \Longrightarrow |T \cap \delta^-(S)| = 1$, and
2. $a \in T \Longrightarrow \sum_{S : a \in \delta^-(S)} y_S = c_a$.

The algorithm will proceed in 2 phases. In the first phase, it will construct a dual feasible solution y and a set F of arcs which has a directed path from the root to every vertex. This may not be an r-arborescence as there might be too many arcs. The arcs in F will satisfy condition 2 above (but not condition 1). In the second phase, the algorithm will remove unnecessary arcs, and will get an r-arborescence satisfying condition 1.

Phase 1 is initialized with $F = \emptyset$ and $y_S = 0$ for all S. While F does not contain a directed path to every vertex in V, the algorithm selects a set S such that (i) inside S, F is strongly connected (i.e. every vertex can reach every vertex) and (ii) $F \cap \delta^-(S) = \emptyset$. This set S exists since we can contract all strongly connected components and in the resulting acyclic digraph, there must be a vertex (which may be coming from the shrinking of a strongly connected component) with no incoming arc (otherwise tracing back from that vertex we would either get to the root or discover a new directed cycle (which we could shrink)). Now we increase y_S as much as possible until a new inequality, say for arc a_k, $\sum_{S : a_k \in \delta^-(S)} y_S \leq c_{a_k}$ becomes an equality. In so doing, the solution y remains dual feasible and still satisfies condition 2. We can now add a_k to F without violating complementary slackness condition 2, and then

we increment k (which at the start we initialized at $k = 1$). And we continue by selecting another set S, and so on, until every vertex is reachable from r in F. We have now such a set $F = \{a_1, a_2, \cdots, a_k\}$ and a dual feasible solution y satisfying condition 2.

In step 2, we eliminate as many arcs as possible, but we consider them in *reverse order* they were added to F. Thus, we let i go from k to 1, and if $F \setminus \{a_i\}$ still contains a directed path from r to every vertex, we remove a_i from F, and continue. We then output the resulting set T of arcs.

The first claim is that T is an arborescence. Indeed, we claim it has exactly $|V| - 1$ arcs with precisely one arc incoming to every vertex $v \in V \setminus \{r\}$. Indeed, if not, there would be two arcs a_i and a_j incoming to some vertex v; say that $i < j$. In the reverse delete step, we should have removed a_j; indeed any vertex reachable from r through a_j could be reached through a_i as well (unless a_i is unnecessary in which case we could get rid of a_i later on).

The second (and final) claim is that the complementary slackness condition 1 is also satisfied. Indeed, assume not, and assume that we have a set S with $y_S > 0$ and $|T \cap \delta^-(S)| > 1$. S was chosen at some point by the algorithm and at that time we added $a_k \in \delta^-(S)$ to F. As there were no other arcs in $\delta^-(S)$ prior to adding a_k to F, it means that all other arcs in $T \cap \delta^-(S)$ must be of the form a_j with $j > k$. In addition, when S was chosen, F was already strongly connected within S; this means that from any vertex inside S, one can go to any other vertex inside S using arcs a_i with $i < k$. We claim that when a_j was considered for removal, it should have been removed. Indeed, assume that a_j is needed to go to vertex v, and that along the path P to v the last vertex in S is $w \in S$. Then we could go to v by using a_k which leads somewhere in S then take arcs a_i with $i < k$ (none of which have been removed yet as $i < k < j$) to $w \in S$ and then continue along path P. So a_j was not really necessary and should have been removed. This shows that complementary slackness condition 1 is also satisfied and hence the arborescence built is optimal.

6.5 Matroid Union

From any matroid $M = (E, \mathcal{I})$, one can construct a dual matroid $M^* = (E, \mathcal{I}^*)$.

Theorem 6.7 *Let $\mathcal{I}^* = \{X \subseteq E : E \setminus X$ contains a base of $M\}$. Then $M^* = (E, \mathcal{I}^*)$ is a matroid with rank function*

$$r_{M^*}(X) = |X| + r_M(E \setminus X) - r_M(E).$$

There are several ways to show this. One is to first show that indeed the size of the largest subset of X in $\mathcal{I}^*$ has cardinality $|X| + r_M(E \setminus X) - r_M(E)$ and then show that r_{M^*} satisfies the three conditions that a rank function of a matroid needs to satisfy (the third one, submodularity, follows from the submodularity of the rank function for M).

One can use Theorem 6.7 and matroid intersection to get a good characterization of when a graph $G = (V, E)$ has two edge-disjoint spanning trees. Indeed, letting M be the graphic matroid of the graph G, we get that G has two edge-disjoint spanning trees if and only if

$$\max_{S \in \mathcal{I} \cap \mathcal{I}^*} |S| = |V| - 1.$$

For the graphic matroid, we know that $r_M(F) = n - \kappa(F)$ where $n = |V|$ and $\kappa(F)$ denotes the number of connected components of (V, F). But by the matroid intersection theorem, we can write:

$$\begin{aligned}\max_{S \in \mathcal{I} \cap \mathcal{I}^*} |S| &= \min_{E_1 \subseteq E} [r_M(E_1) + r_{M^*}(E \setminus E_1)] \\ &= \min_{E_1 \subseteq E} [(n - \kappa(E_1)) + (|E \setminus E_1| + \kappa(E) - \kappa(E_1))] \\ &= \min_{E_1 \subseteq E} [n + 1 + |E \setminus E_1| - 2\kappa(E_1)],\end{aligned}$$

where we replaced $\kappa(E)$ by 1 since otherwise G would even have one spanning tree. Rearranging terms, we get that G has two edge-dsjoint spanning trees if and only if for all $E_1 \subseteq E$, we have that $E \setminus E_1 \geq 2(\kappa(E_1) - 1)$. If this inequality is violated for some E_1, we can add to E_1 any edge that does not decrease $\kappa(E_1)$. In other words, if the connected components of E_1 are $V_1, V_2, \cdots, V_p$ then we can assume that $E_1 = E \setminus \delta(V_1, V_2, \cdots V_p)$ where $\delta(V_1, \cdots, V_p) = \{(u, v) \in E : u \in V_i, v \in V_j \text{ and } i \neq j\}$. Thus we have shown:

Theorem 6.8 *G has two edge-disjoint spanning trees if and only if for all partitions V_1, $V_2, \cdots V_p$ of V, we have*

$$|\delta(V_1, \cdots, V_p)| \geq 2(p - 1).$$

Theorem 6.8 can be generalized to an arbitrary number of edge-disjoint spanning trees. This result is not proved here.

Theorem 6.9 *G has k edge-disjoint spanning trees if and only if for all partitions V_1, $V_2, \cdots V_p$ of V, we have*

$$|\delta(V_1, \cdots, V_p)| \geq k(p - 1).$$

From two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$, we can also define its union by $M_1 \cup M_2 = (E, \mathcal{I})$ where $\mathcal{I} = \{S_1 \cup S_2 : S_1 \in \mathcal{I}_1, S_2 \in \mathcal{I}_2\}$. Notice that we do not impose the two matroids to be identical as we just did for edge-disjoint spanning trees.

We can show that:

Theorem 6.10 (Matroid Union) *$M_1 \cup M_2$ is a matroid. Furthermore its rank function is given by*

$$r_{M_1 \cup M_2}(S) = \min_{F \subseteq S} \{|S \setminus F| + r_{M_1}(F) + r_{M_2}(F)\}.$$

Proof: To show that it is a matroid, assume that $X, Y \in \mathcal{I}$ with $|X| < |Y|$. Let $X = X_1 \cup X_2$ and $Y = Y_1 \cup Y_2$ where $X_1, Y_1 \in \mathcal{I}_1$ and $X_2, Y_2 \in \mathcal{I}_2$. We can furthermore assume that the X_i's are disjoint and so are the Y_i's. Finally we assume that among all choices for X_1, X_2, Y_1 and Y_2, we choose the one maximizing $|X_1 \cap Y_1| + |X_2 \cap Y_2|$. Since $|Y| > |X|$, we can assume that $|Y_1| > |X_1|$. Thus, there exists $e \in (Y_1 \setminus X_1)$ such that $X_1 \cup \{e\}$ is independent for M_1. The maximality implies that $e \notin X_2$ (otherwise consider $X_1 \cup \{e\}$ and $X_2 \setminus \{e\}$). But this implies that $X \cup \{e\} \in \mathcal{I}$ as desired.

We now show the expression for the rank function. The fact that it is $\leq$ is obvious as an independent set $S \in \mathcal{I}$ has size $|S \setminus F| + |S \cap F| \leq |S \setminus F| + r_{M_1}(F) + r_{M_2}(F)$ and this is true for any F.

For the converse, let us prove it for the entire ground set $S = E$. Once we prove that

$$r_{M_1 \cup M_2}(E) = \min_{F \subseteq S} \{|E \setminus F| + r_{M_1}(F) + r_{M_2}(F)\},$$

the corresponding statement for any set S will follow by just restricting our matroids to S.

Let X be a base of $M_1 \cup M_2$. The fact that $X \in \mathcal{I}$ means that $X = X_1 \cup X_2$ with $X_1 \in \mathcal{I}_1$ and $X_2 \in \mathcal{I}_2$. We can furthermore assume that X_1 and X_2 are disjoint and that $r_{M_2}(X_2) = r_{M_2}(E)$ (otherwise add elements to X_2 and possibly remove them from X_1). Thus we can assume that $|X| = |X_1| + r_{M_2}(E)$. We have that $X_1 \in \mathcal{I}_1$ and also that X_1 is independent for the dual of M_2 (as the complement of X_1 contains a base of M_2). In other words, $X_1 \in \mathcal{I}_1 \cap \mathcal{I}_2^*$. The proof is completed by using the matroid intersection theorem and Theorem 6.7:

$$\begin{aligned} r_{M_1 \cup M_2}(E) = |X| &= \max_{X_1 \in \mathcal{I}_1 \cap \mathcal{I}_2^*} (|X_1| + r_{M_2}(E)) \\ &= \min_{E_1 \subseteq E} \left(r_{M_1}(E_1) + r_{M_2^*}(E \setminus E_1) + r_{M_2}(E)\right) \\ &= \min_{E_1 \subseteq E} (r_{M_1}(E_1) + |E \setminus E_1| + r_{M_2}(E_1) - r_{M_2}(E) + r_{M_2}(E)) \\ &= \min_{E_1 \subseteq E} (|E \setminus E_1| + r_{M_1}(E_1) + r_{M_2}(E_1)), \end{aligned}$$

as desired. △

Since Theorem 6.10 says that $M_1 \cup M_2$ is a matroid, we know that its rank function is submodular. This is, however, not obvious from the formula given in the theorem.

6.5.1 Spanning Tree Game

The spanning tree game is a 2-player game. Each player in turn selects an edge. Player 1 starts by deleting an edge, and then player 2 fixes an edge (which has not been deleted yet); an edge fixed cannot be deleted later on by the other player. Player 2 wins if he succeeds in constructing a spanning tree of the graph; otherwise, player 1 wins. The question is which graphs admit a *winning strategy* for player 1 (no matter what the other player does), and which admit a winning strategy for player 2.

Theorem 6.11 *For the spanning tree game on a graph $G = (V, E)$, player 1 has a winning strategy if and only if G does not have two edge-disjoint spanning trees. Otherwise, player 2 has a winning strategy.*

If G does not have 2 edge-disjoint spanning trees then, by Theorem 6.8, we know that there exists a partition $V_1, \cdots, V_p$ of V with $|\delta(V_1, \cdots, V_p)| \leq 2(p-1) - 1$. The winning strategy for player 1 is then to always delete an edge from $\delta(V_1, \cdots, V_p)$. As player 1 plays before player 2, the edges in $\delta(V_1, \cdots, V_p)$ will be exhausted before player 2 can fix $p - 1$ of them, and therefore player 2 loses. The converse is the subject of exercise 6-4.

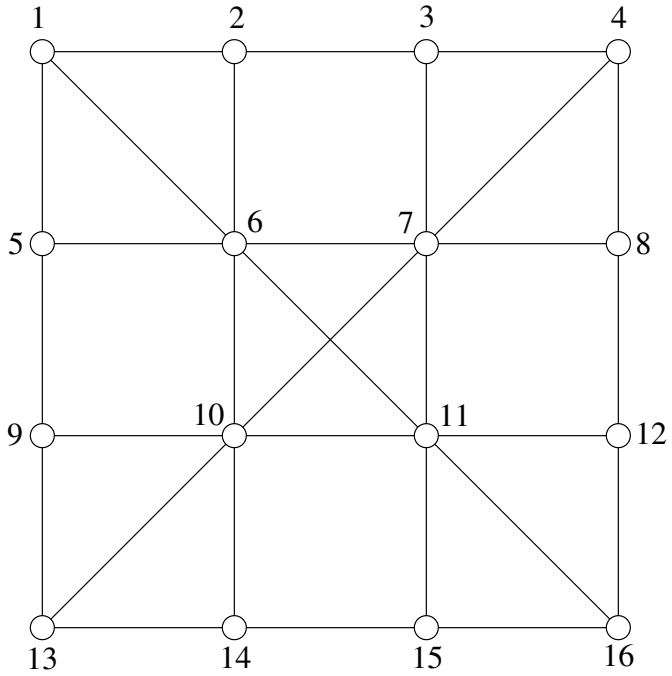
Exercise 6-2. Derive from theorem 6.10 that the union of k matroids M_1, $M_2, \cdots, M_k$ is a matroid with rank function

$$r_{M_1 \cup M_2 \cup \cdots \cup M_k}(S) = \min_{F \subseteq S} \{|S \setminus F| + r_{M_1}(F) + r_{M_2}(F) + \cdots + r_{M_k}(F)\}.$$

Exercise 6-3. Derive Theorem 6.9 from Exercise 6-2.

Exercise 6-4. Assume that G has 2 edge-disjoint spanning trees. Give a winning strategy for player 2 in the spanning tree game.

Exercise 6-5. Find two edge-disjoint spanning trees in the following graph with 16 vertices and 30 edges or prove that no such trees exist.



7. Lecture notes on the ellipsoid algorithm

The simplex algorithm was the first algorithm proposed for linear programming, and although the algorithm is quite fast in practice, no variant of it is known to be polynomial time. The Ellipsoid algorithm is the first polynomial-time algorithm discovered for linear programming. The Ellipsoid algorithm was proposed by the Russian mathematician Shor in 1977 for general convex optimization problems, and applied to linear programming by Khachyan in 1979. Contrary to the simplex algorithm, the ellipsoid algorithm is not very fast in practice; however, its theoretical polynomiality has important consequences for combinatorial optimization problems as we will see. Another polynomial-time algorithm, or family of algorithms to be more precise, for linear programming is the class of interior-point algorithms that started with Karmarkar's algorithm in 1984; interior-point algorithms are also quite practical but they do not have the same important implications to combinatorial optimization as the ellipsoid algorithm does.

The problem being considered by the ellipsoid algorithm is:

Given a bounded convex set $P \in \mathbb{R}^n$ find $x \in P$.

We will see that we can reduce linear programming to finding an x in $P = \{x \in \mathbb{R}^n : Cx \leq d\}$.

The ellipsoid algorithm works as follows. We start with a big ellipsoid E that is guaranteed to contain P. We then check if the center of the ellipsoid is in P. If it is, we are done, we found a point in P. Otherwise, we find an inequality $c^T x \leq d_i$ which is satisfied by all points in P (for example, it is explicitly given in the description of P) which is not satisfied by our center. One iteration of the ellipsoid algorithm is illustrated in Figure 7.1. The ellipsoid algorithm is the following.

- Let E_0 be an ellipsoid containing P
- while center a_k of E_k is not in P do:
  - Let $c^T x \leq c^T a_k$ be such that $\{x : c^T x \leq c^T a_k\} \supseteq P$
  - Let E_{k+1} be the minimum volume ellipsoid containing $E_k \cap \{x : c^T x \leq c^T a_k\}$
  - $k \leftarrow k + 1$

The ellipsoid algorithm has the important property that the ellipsoids contructed shrink in volume as the algorithm proceeds; this is stated precisely in the next lemma. This means that if the set P has positive volume, we will eventually find a point in P. We will need to deal with the case when P has no volume (i.e. P has just a single point), and also discuss when we can stop and be guaranteed that either we have a point in P or we know that P is empty.

Lemma 7.1 $\frac{Vol(E_{k+1})}{Vol(E_k)} < e^{-\frac{1}{2(n+1)}}$.

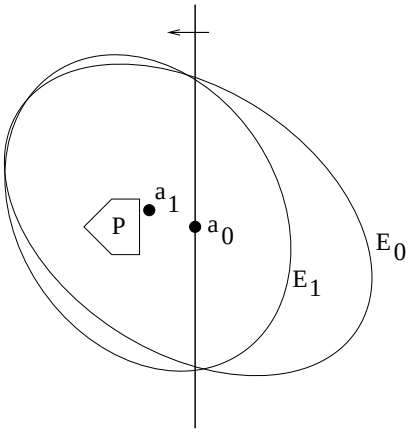


Figure 7.1: One iteration of the ellipsoid algorithm.

Before we can state the algorithm more precisely, we need to define ellipsoids.

Definition 7.1 *Given a center a, and a positive definite matrix A, the ellipsoid $E(a, A)$ is defined as $\{x \in \mathbb{R}^n : (x-a)^T A^{-1}(x-a) \leq 1\}$.*

One important fact about a positive definite matrix A is that there exists B such that $A = B^T B$, and hence $A^{-1} = B^{-1}(B^{-1})^T$. Ellipsoids are in fact just affine transformations of unit spheres. To see this, consider the (bijective) affine transformation $T : x \to y = (B^{-1})^T(x-a)$. It maps $E(a, A) \to \{y : y^T y \leq 1\} = E(0, I)$.

We first consider the simple case in which the ellipsoid E_k is the unit sphere and the inequality we generate is $x_1 \geq 0$. We claim that the ellipsoid containing $E_k \cap \{x : x_1 \geq 0\}$ is

$$E_{k+1} = \left\{ x : \left(\frac{n+1}{n}\right)^2 \left(x - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2} \sum_{i=2}^{n} x_i^2 \leq 1 \right\}.$$

Indeed, if we consider an $x \in E_k \cap \{x : x_1 \geq 0\}$, we see that

$$\begin{aligned}
&\left(\frac{n+1}{n}\right)^2 \left(x - \frac{1}{n+1}\right)^2 + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2 \\
&= \frac{n^2+2n+1}{n^2}x_1^2 - \left(\frac{n+1}{n}\right)^2 \frac{2x_1}{n+1} + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=2}^{n} x_i^2 \\
&= \frac{2n+2}{n^2}x_1^2 - \frac{2n+2}{n^2}x_1 + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=1}^{n} x_i^2 \\
&= \frac{2n+2}{n^2}x_1(x_1-1) + \frac{1}{n^2} + \frac{n^2-1}{n^2}\sum_{i=1}^{n} x_i^2 \\
&\leq \frac{1}{n^2} + \frac{n^2-1}{n^2} \leq 1.
\end{aligned}$$

In this specific case, we can prove easily lemma 7.1.

Proof: The volume of an ellipsoid is proportional to the product of its side lengths. Hence the ratio between the unit ellipsoid E_k and E_{k+1} is

$$\begin{aligned}
\frac{Vol(E_{k+1})}{VolE_k} &= \frac{(\frac{n}{n+1})(\frac{n^2}{n^2-1})^{\frac{n-1}{2}}}{1} \\
&= \left(\frac{n}{n+1}\right)\left(\frac{n^2}{n^2-1}\right)^{\frac{n-1}{2}} \\
&< e^{-\frac{1}{n+1}} e^{\frac{n-1}{(n^2-1)2}} = e^{-\frac{1}{n+1}} e^{\frac{1}{2(n+1)}} = e^{-\frac{1}{2(n+1)}},
\end{aligned}$$

where we have used the fact that $1 + x \leq e^x$ for all x, with strict inequality if $x \neq 0$. $\triangle$

Now consider the slightly more general case in which the ellipsoid is also the unit sphere but we have an arbitrary constraint $d^T x \leq 0$. We want to find an ellipsoid that contains $E(0, I) \cap \{x : d^T x \leq 0\}$ (we let $\|d\| = 1$; this can be done by scaling both sides), it is easy to verify that we can take $E_{k+1} = E(-\frac{1}{n+1}d, F)$, where $F = \frac{n^2}{n^2-1}(I - \frac{2}{n+1}dd^T)$, and the ratio of the volumes is $\leq \exp\left(-\frac{1}{2(n+1)}\right)$.

Now we deal with the case where E_k is not the unit sphere. We take advantage of the fact that linear transformations preserve ratios of volumes.

$$\begin{array}{ccc} E_k & \stackrel{T}{\rightarrow} & E(0,1) \\ & & \downarrow \\ E_{k+1} & \stackrel{T^{-1}}{\leftarrow} & E' \end{array} \tag{1}$$

Let a_k be the center of E_k, and $c^T x \leq c^T a_k$ be the halfspace through a_k that contains P. Therefore, the half-ellipsoid that we are trying to contain is $E(a_k, A) \cap \{x : c^T x \leq c^T a_k\}$.

Let's see what happens to this half-ellipsoid after the transformation T defined by $y = T(x) = (B^{-1})^T(x-a)$. This transformation transforms $E_k = E(a_k, A)$ to $E(0, I)$. Also,

$$\{x : c^T x \leq c^T a_k\} \xrightarrow{T} \{y : c^T(a_k + B^T y) \leq c^T a_k\} = \{y : c^T B^T y \leq 0\} = \{y : d^T x \leq 0\}, \quad (2)$$

where d is given by the following equation:

$$d = \frac{Bc}{\sqrt{c^T B^T B c}} = \frac{Bc}{\sqrt{c^T A c}}. \quad (3)$$

Let $b = B^T d = \frac{Ac}{\sqrt{c^T A c}}$. This implies:

$$E_{k+1} = E\left(a_k - \frac{1}{n+1}b, \frac{n^2}{n^2-1}B^T\left(I - \frac{2}{n+1}dd^T\right)B\right) \quad (4)$$

$$= E\left(a_k - \frac{1}{n+1}b, \frac{n^2}{n^2-1}\left(A - \frac{2}{n+1}bb^T\right)\right). \quad (5)$$

To summarize, here is the Ellipsoid Algorithm:

1. Start with $k = 0$, $E_0 = E(a_0, A_0) \supseteq P$, $P = \{x : Cx \leq d\}$.
2. While $a_k \notin P$ do:
   - Let $c^T x \leq d$ be an inequality that is valid for all $x \in P$ but $c^T a_k > d$.
   - Let $b = \frac{A_k c}{\sqrt{c^T A_k c}}$.
   - Let $a_{k+1} = a_k - \frac{1}{n+1}b$.
   - Let $A_{k+1} = \frac{n^2}{n^2-1}(A_k - \frac{2}{n+1}bb^T)$.

Claim 7.2 $\frac{Vol(E_{k+1})}{Vol(E_k)} < \exp\left(-\frac{1}{2(n+1)}\right)$.

After k iterations, $Vol(E_k) \leq Vol(E_0)\exp\left(-\frac{k}{2(n+1)}\right)$. If P is nonempty then the Ellipsoid Algorithm should find $x \in P$ in at most $2(n+1)\ln\frac{Vol(E_0)}{Vol(P)}$ steps.

In general, for a full-dimensional polyhedron described as $P = \{x : Cx \leq d\}$, one can show that $\frac{Vol(E_0)}{Vol(P)}$ is polynomial in the encoding length for C and d. We will not show this in general, but we will focus on the most important situation in combinatorial optimization when we are given a set $S \subseteq \{0,1\}^n$ (not explicitly, but for example as the incidence vectors of all matchings in a graph) and we would like to optimize over $P = conv(S)$. We will make the assumption that P is full-dimensional; otherwise, one can eliminate one or several variables and obtain a smaller full-dimensional problem.

From feasibility to Optimization First, let us show how to reduce such an optimization problem to the problem of finding a feasible point in a polytope. Let $c^T x$ with $c \in \mathbb{R}^n$ be our objective function we would like to minimize over P. Assume without loss of generality that $c \in \mathbb{Z}^n$. Instead of optimizing, we can check the non-emptiness of

$$P' = P \cap \{x : c^T x \leq d + \frac{1}{2}\}$$

for $d \in \mathbb{Z}$ and our optimum value corresponds to the smallest such d. As $S \subseteq \{0,1\}^n$, d must range in $[-nc_{max}, nc_{max}]$ where $c_{max} = \max_i c_i$. To find d, we can use binary search (and check the non-emptiness of P' with the ellipsoid algorithm). This will take $O(\log(nc_{max})) = O(\log n + \log c_{max})$ steps, which is polynomial.

Starting Ellipsoid. Now, we need to consider using the ellipsoid to find a feasible point in P' or decide that P' is empty. As starting ellipsoid, we can use the ball centered at the vector $(\frac{1}{2}, \frac{1}{2}, \cdots, \frac{1}{2})$ and of radius $\frac{1}{2}\sqrt{n}$ (which goes through all $\{0,1\}^n$ vectors). This ball has volume $Vol(E_0) = \frac{1}{2^n}(\sqrt{n})^n Vol(B_n)$, where B_n is the unit ball. We have that $Vol(B_n) = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)}$, which for the purpose here we can even use the (very weak) upper bound of $\pi^{n/2}$ (or even 2^n). This shows that $\log(Vol(E_0)) = O(n \log n)$.

Termination Criterion. We will now argue that if P' is non-empty, its volume is not too small. Assume that P' is non-empty, say $v_0 \in P' \cap \{0,1\}^n$. Our assumption that P is full-dimensional implies that there exists $v_1, v_2, \cdots, v_n \in P \cap \{0,1\}^n = S$ such that the "simplex" $v_0, v_1, \cdots, v_n$ is full-dimensional. The v_i's may not be in P'. Instead, define w_i for $i = 1, \cdots, n$ by:

$$w_i = \begin{cases} v_i & \text{if } c^T v_i \leq d + \frac{1}{2} \\ v_0 + \alpha(v_i - v_0) & \text{otherwise} \end{cases}$$

where $\alpha = \frac{1}{2nc_{max}}$. This implies that $w_i \in P'$ as

$$c^T w_i = c^T v_0 + \alpha c^T (v_i - v_0) \leq d + \frac{1}{2nc_{max}} nc_{max} = d + \frac{1}{2}.$$

We have that P' contains $C = conv(\{v_0, w_1, w_2, \cdots, w_n\})$ and $Vol(C)$ is $\frac{1}{n!}$ times the volume of the parallelipiped spanned by $w_i - v_0 = \beta_i(v_i - v_0)$ (with $\beta_i \in \{\alpha, 1\}$) for $i = 1, \cdots, n$. This paralleliped has volume equal to the product of the β_i (which is at least α^n) times the volume of a parallelipiped with integer vertices, which is at least 1. Thus,

$$Vol(P') \geq Vol(C) = \frac{1}{n!}\left(\frac{1}{2nc_{max}}\right)^n.$$

Taking logs, we see that the number of iterations of the ellipsoid algorithm before either discovering that P' is empty or a feasible point is at most

$$\log(Vol(E_0)) - \log(Vol(P')) = O(n \log n + n \log c_{max}).$$

This is polynomial.

Separation Oracle. To run the ellipsoid algorithm, we need to be able to decide, given $x \in \mathbb{R}^n$, whether $x \in P'$ or find a violated inequality. The beauty here is that we do not necessarily need a complete and explicit description of P in terms of linear inequalities. We will see examples in which we can even apply this to exponential-sized descriptions. What we need is a *separation oracle* for P: Given $x^* \in \mathbb{R}^n$, either decide that $x^* \in P$ or find an inequality $a^T x \leq b$ valid for P such that $a^T x^* > b$. If this separation oracle runs in polynomial-time, we have succeeded in finding the optimum value d when optimizing $c^T x$ over P (or S).

Finding an optimum solution. There is one more issue. This algorithm gives us a point x^* in P of value at most $d + \frac{1}{2}$ where d is the optimum value. However, we are interested in finding a point $x \in P \cap \{0,1\}^n = S$ of value exactly d. This can be done by starting from x^* and finding *any extreme* point x of P such that $c^T x \leq c^T x^*$. Details are omitted.

In summary, we obtain the following important theorem shown by Grötschel, Lovász and Schrijver, 1979.

Theorem 7.3 *Let $S = \{0,1\}^n$ and $P = conv(S)$. Assume that P is full-dimensional and we are given a separation oracle for P. Then, given $c \in \mathbb{Z}^n$, one can find $\min\{c^T x : x \in S\}$ by the ellipsoid algorithm by using a polynomial number of operations and calls to the separation oracle.*

To be more precise, the number of iterations of the ellipsoid algorithm for the above application is $O(n^2 \log^2 n + n^2 \log^2 c_{max})$, each iteration requiring a call to the separation oracle and a polynomial number of operations (rank-1 updates of the matrix A, etc.).

Here are two brief descriptions of combinatorial optimization problems that can be solved with this approach.

Minimum Cost Arborescence Problem. We have seen a combinatorial algorithm to solve the minimum cost arborescence problem, and it relied on solving by a primal-dual method the linear program:

$$LP = \min \sum_{a \in A} c_a x_a$$

subject to:

$$(P) \qquad \sum_{a \in \delta^-(S)} x_a \geq 1 \qquad \forall S \subseteq V \setminus \{r\}$$

$$x_a \geq 0 \qquad a \in A.$$

Instead, we could use the ellipsoid algorithm to directly solve this linear program in polynomial-time. Indeed, even though this linear program has an exponential number of constraints (in the size of the graph), a separation oracle for it can be easily defined. Indeed consider x^*. If $x^*_a < 0$ for some $a \in A$, just return the inequality $x_a \geq 0$. Otherwise, for every $t \in V \setminus \{r\}$,

consider the minimum $r-t$ cut problem in which the capacity on arc a is given by x^*_a. As we have seen, this can be solved by maximum flow computations. If for some $t \in V \setminus \{r\}$, the minimum $r-t$ cut has value less than 1 then we have found a violated inequality by x^*. Otherwise, we have that $x^* \in P$. Our separation oracle can thus be implemented by doing $|V|-1$ maximum flow computations, and hence is polynomial. We are done.

Maximum Weight Matching Problem. For the maximum weight matching problem in a general graph $G=(V,E)$, Edmonds has shown that the convex hull of all matchings is given by:

$$\begin{aligned} \sum_{e\in E(S)} x_e &\le \frac{|S|-1}{2} && S \subset V, |S| \text{ odd} \\ \sum_{e\in\delta(v)} x_e &\le 1 && v \in V \\ x_e &\ge 0 && e \in E. \end{aligned} \tag{6}$$

Given x^*, we can easily check if x^* is nonnegative and if the n constraints $\sum_{e\in\delta(v)} x_e \le 1$ are satisfied. There is an exponential number of remaining constraints, but we will show now that they can be checked by doing a sequence of minimum cut computations.

Assume $x \ge 0$ satisfies $\sum_{e\in\delta(v)} x_e \le 1$ for every $v \in V$, and we would like to decide if

$$\sum_{e\in E(S)} x_e \le \frac{|S|-1}{2}$$

for all $S \subset V$, $|S|$ odd, and if not produce such a violated set S. We can assume without loss of generality that $|V|$ is even (otherwise, simply add a vertex). Let

$$s_v = 1 - \sum_{e\in\delta(v)} x_e,$$

for all $v \in V$. Observe that $\sum_{e\in E(S)} x_e \le \frac{|S|-1}{2}$ is equivalent to

$$\sum_{v\in S} s_v + \sum_{e\in\delta(S)} x_e \ge 1.$$

Define a new graph H, whose vertex set is $V \cup \{u\}$ (where u is a new vertex) and whose edge set is $E \cup \{(u,v) : v \in V\}$. In this new graph, let the capacity u_e of an edge be x_e if $e \in E$ or s_v for $e=(u,v)$. Observe that in this graph,

$$\sum_{v\in S} s_v + \sum_{e\in\delta(S)} x_e \ge 1$$

if and only if

$$\sum_{e\in\delta_H(S)} u_e \ge 1,$$

where $\delta_H(S)$ are the edges of H with one endpoint in S. Thus, to solve our separation problem, it is enough to be able to find the minimum cut in H among all the cuts defined by sets S with $S \subseteq V$, $|S|$ odd. This is a special case of the minimum T-odd cut problem seen previously. If the minimum T-odd cut problem returns a minimum cut of value greater or equal to 1, we know that all inequalities are satisfied; otherwise, a minimum T-odd cut provides a violated inequality.