Combinatorial Optimization Algorithms

Lecture Notes by László Lovász
Based on classroom notes of Claire Kenyon

1. Introduction: Examples and models

   Shortest paths and trees — Assignment and matching — Travelling Salesman — Max cut — Stable sets, blocking sets, and colorations — Bin packing, subset sum and knapsack

2. The greedy algorithm

   Kruskal's algorithm — An introduction to matroids — Greedy heuristic — Rank quotients — Greedy blocking set — Greedy travelling salesman tours — Bin packing — The knapsack problem

3. Local improvement

   Exchange heuristics for the Traveling Salesman Problem — Maximum cuts — Aiming at local optima — Randomized exchange — Simulated annealing

4. Enumeration and branch-and-bound

5. Dynamic programming

   The subset-sum problem — Minimal triangulation of a convex polygon — Steiner trees in planar graphs — Optimization on tree-like graphs

6. Augmenting paths

   Assignments and bipartite matching — Non-bipartite matching — Matroid intersection — Max-flow-min-cut — Minimum cost flow.

7. Uncrossing

8. Linear programming

   Linear inequalities and polyhedra — Farkas' Lemma and the Duality theorem — Fourier–Motzkin elimination — Simplex method — Ellipsoid method — interior-point methods

   Polyhedral combinatorics — Matching polytope — Stable set polytope — Perfect graphs — Matroid polytope

   Linear programs in combinatorial optimization algorithms — Separation — Cutting plane generation — Branch-and-Cut

Linear programming in heuristics — A polynomial approximation scheme for bin packing — Blocking sets in hypergraphs with small Vapnik-Červonenkis dimension

9. Changing the objective function

Kruskal's algorithm revisited — Minimum weight perfect matching in a bipartite graph — Optimum arborescence — Scaling I: From pseudopolynomial to polynomial — Scaling II: From polynomial to strongly polynomial — Scaling III: Heuristics — Lagrangean relaxation

10. Matrix methods

Matrix inversion in parallel — Determinants and matchings — Determinants and connectivity — Semidefinite optimization

Recommended books:

A. Bachem, M. Grötschel, B. Korte (1983): *Mathematical Programming: the State of the Art*, Springer-Verlag.

M. Grötschel, L. Lovász and A. Schrijver (1988): *Geometric Algorithms and Combinatorial Optimization*, Springer, 1988.

B. Korte, L. Lovász, H. J. Prömel, A. Schrijver, *Paths, flows, and VLSI-Layout*, Springer, 235–265.

G. L. Nemhauser and L. A. Wolsey (1988): *Integer and Combinatorial Optimization*, Wiley–Interscience.

C.H. Papadimitriou and K. Steiglitz (1982): *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

A. Schrijver (1986): *Theory of Integer and Linear Programming*, Wiley–Interscience.

1 Introduction

Optimizing means finding the maximum or minimum of a certain function, defined on some domain. Classical theories of optimization (differential calculus, variational calculus, optimal control theory) deal with the case when this domain is infinite. From this angle, the subject of combinatorial optimization, where the domain is typically finite, might seem trivial: it is easy to say that “we choose the best from this finite number of possibilities”. But, of course, these possibilities may include all trees on n nodes, or all Hamilton circuits of a complete graph, and then listing all possibilities to find the best among them is practically hopeless even for instances of very moderate size. In the framework of complexity theory (Chapter 29), we want to find the optimum in polynomial time. For this (or indeed, to do better than listing all solutions) the special structure of the domain must be exploited. Often when the objective function is too wild, the constraints too complicated, or the problem size too large, it is impossible to find an optimum solution. This is quite often not just a practical experience; mathematics and computer science have developed theories to make intuitive assertions about the difficulty of certain problems precise. Foremost of these is the theory of NP-completeness. In cases when optimum solutions are too hard to find, algorithms (so-called *heuristics*) can often be designed that produce approximately optimal solutions. It is important that these suboptimal solutions have a guaranteed quality; e.g., for a given maximization problem, the value of the heuristic solution is at least 90% of the optimum value for every input. While not so apparent on the surface, of equal importance are algorithms, called *dual heuristics*, that provide (say for a maximization problem again) upper bounds on the optimum value. Since dual heuristics are usually obtained by dropping or relaxing the constraints so as to make the problem easier to solve, they are also called *relaxations*. These bounds are important in the analysis of heuristics, since (both theoretically and in practice) one compares the value obtained by a heuristic with the value obtained by the dual heuristic (instead of the true optimum, which is unknown). Relaxations also play an important role in general solution schemes for hard problems like branch-and-bound. There are basically two ways of presenting “combinatorial optimization”: by problems (models) or by methods. In the introductory part we survey some problems, but for the rest we organize our material according to the second approach. We will describe the fundamental algorithmic techniques in detail and illustrate them on problems to which these methods have been applied successfully. Some important aspects of combinatorial optimization algorithms we can only touch. One such aspect is *parallelism*. There is no doubt that the computers of the future will be parallel machines. A systematic treatment of parallel algorithms is difficult since there are many computer architectures, based on different principles, and each architecture leads to a different model of parallel computational complexity. Another important aspect is the *online* solution of combinatorial problems. We treat here "static" problems, where all data are known before the optimization algorithm is called. In many practical situations, however, data come in one by one, and decisions must be made before the next piece of data arrives. The theoretical modelling of such situations is difficult, and we refrain from discussing the many possibilities. A third disclaimer of this type is that we focus on the *worst-case* analysis of algorithms. From a practical point of view, *average-case analysis*, i.e. the analysis of algorithms on random input, would be more important; but for a mathematical treatment of this, one has to make assumptions about the input distribution, and except for very simple cases, such assumptions are extremely difficult to justify and lead to unresolvable controversies. Finally, we should mention the increasing significance of *randomization.* (This should not be confused with the average case analysis of algorithms.) It is often used in conjunction with determinant methods, but is an important tool in avoiding "traps" (degeneracies), in reaching tricky "corners" of the domain, and in many other situations. Combinatorial optimization problems typically have as inputs

both numbers and combinatorial structures (e.g. a graph with weights on the edges). In the Turing machine model, both are encoded as 0-1 strings; but in the RAM machine model it is natural to consider the input as a sequence of integers. If the input also involves a combinatorial structure, then the combinatorial structure can be considered as a set of 0's and 1's. We denote by $\langle a \rangle$ the number of bits in the binary representation of the integer a; for a matrix $A = (a_{ij})$ of integers, we define $\langle A \rangle = \sum \langle a_{ij} \rangle$. An algorithm (with input integers $a_1, \ldots, a_n$) is *polynomial* if it can be implemented on the RAM machine so that the number of bit-operations performed is bounded by a polynomial of the number of input bits $\langle a_1 \rangle + \ldots + \langle a_n \rangle$. Considering the input as a set of numbers allows two important versions of this notion. An algorithm (with input integers $a_1, \ldots, a_n$) is *pseudopolynomial*, if it can be implemented on the RAM machine so that the number of bit-operations performed is bounded by a polynomial of $|a_1| + \ldots + |a_n|$. (This can also be defined on the Turing machine model: it corresponds to encoding an integer a by a string of a 1's. Thus a pseudopolynomial algorithm is also called *polynomial in the unary encoding.*) Clearly every polynomial algorithm is pseudopolynomial, but not the other way around: testing primality in the trivial way by searching through all smaller integers is pseudopolynomial but not polynomial. An algorithm (with input integers $a_1, \dots, a_n$) is *strongly polynomial* if it can be implemented on the RAM machine in $O(n^c)$ steps with numbers of $O((\langle a_1 \rangle + \ldots + \langle a_n \rangle)^c)$ digits for some $c > 0$. Clearly every strongly polynomial algorithm is polynomial, but not the other way around: e.g. the euclidean algorithm is polynomial but not strongly polynomial. On the other hand, Kruskal's algorithm for shortest spanning tree is strongly polynomial. **Some models.**(To be written.)

2 The greedy algorithm

Kruskal's algorithm. The most natural principle we can try to build an optimization algorithm on is *greediness*: building up the solution by making the best choice locally. As the most important example of an optimization problem where this simple idea works, let us recall Kruskal's Algorithm for a shortest spanning tree from Chapters 2, 9 and 40. Given a graph G with n nodes, and a length function $\ell : E(G) \to \mathbb{Z}$, we want to find a shortest spanning tree (where the length of a subgraph of G is defined as the sum of the lengths of its edges). The algorithm constructs the tree by finding its edges $e_1, e_2, \ldots e_{n-1}$ one by one: — e_1 is an edge of minimum length; — e_k is an edge such that $e_k \notin \{e_1, \ldots, e_{k-1}\}$, $\{e_1, \ldots, e_k\}$ is a forest and

$$c(e_k) = \min\{c(e) \mid e \notin \{e_1, \ldots, e_{k-1}\} \text{ and } \{e_1, \ldots e_{k-1}, e\} \text{ is a forest } \} .$$

It is clear that the algorithm produces a spanniong tree. Each step of the algorithm makes locally the best choice: this is why it is called a *greedy* algorithm. Note that we may have several choices for e_k, and accordingly, the spanning tree constructed by this greedy algorithm is not uniquely determined. Kruskal's Theorem (going back actually to Borůvka in 1926) asserts that $\{e_i : 1 \le i \le n-1\}$ is a shortest spanning tree of G:

Theorem 2.1 *Any spanning tree given by Kruskal's algorithm is a shortest spanning tree.*

Proof. Let T be a spanning tree constructed by Kruskal's algorithm and let S be a spanning tree with minimum length; if S is not unique, we choose S to have as many edges in common with T as possible. Assume that $T \neq S$, and let e_k be the first edge of T not is S. Then $S \cup \{e_k\}$ contains a unique circuit C. The endpoints of e_k are in different connected components of $T \setminus e_k$ (where a node of G not covered by $T \setminus e_k$ is considered as single-node connected component), and hence C must contain another edge f connecting different connected components of $\{e_1, \ldots, e_{k-1}$. Now f was a candidate when e_k was chosen, and hence $c(f) \geq c(e_k)$. But this means that $S \cup \{e_k\} \setminus f$ is a spanning tree that is either shorter than S or has the same length as S but more edges in common with T, a contradiction. ∎

The graph structure plays little role in the algorithm; the only information about the graph used is that "$(e_1, \ldots, e_{k-1}, e)$ is a forest". In fact, this observation is one of the possible routes to the notion of *matroids,* as we shall see later. There are other problems where the greedy algorithm (with an appropriate interpretation) gives an optimum solution. The notion of *greedoids* (see Korte, Lovász and Schrader 1990) is an attempt to describe a general class of such problems. Other examples are polymatroids, certain 0-1 matrices (see Hoffman 1978), coloring of various classes of perfect graphs, etc. In fact, an optimization model where the greedy solution is optimal was described by Monge in the 18th century! **Greedy heuristic.**

But in most optimization problems greed does not pay: the greedy solution is not optimal in general. We may still use, however, a greedy algorithm as a heuristic, to obtain a "reasonable" solution (and in practice it is very often used indeed). We measure the quality of the heuristic by comparing the value of the objective function at the heuristic solution with its value at the optimum solution. To be precise, consider, say, a minimization problem. For a given instance, let v_{opt} denote the optimum objective value, and v_{heur}, the objective value achieved by a given heuristic (if the heuristic includes free choices at certain points, we define v_{heur} as the value achieved by the worst possible choices). We define the *performance ratio* of a heuristic as the supremum of $v_{\text{heur}}/v_{\text{opt}}$ over all problem instances. The *asymptotic performance ratio* is the lim sup of this ratio, assuming that $v_{\text{opt}} \to \infty$. For a maximization problem, we replace this quotient by $v_{\text{opt}}/v_{\text{heur}}$. **Rank quotients.** Let S be a finite set, $c : S \to \mathbb{Z}_+$, a cost function on S, and $\mathcal{F} \subset 2^S$ a hereditary family of subsets of S, i.e., a set of subsets such that $X \in \mathcal{F}$ and $Y \subseteq X$ implies $Y \in \mathcal{F}$. The goal is to find $\max\{c(X) = \sum_{e \in X} c(e) \mid X \in \mathcal{F}\}$. In this more general setting, the greedy algorithm can still be easily formulated: it constructs a maximal set X by finding its elements $e_1, e_2, \ldots$ one by one. — e_1 is the element of maximum cost in $\cup_{X \in \mathcal{F}} X$. — e_k is defined such that $e_k \neq e_1, \ldots, e_{k-1}$, $\{e_1, \ldots, e_k\} \in \mathcal{F}$ and

$$c(e_k) = \max\{c(e) \mid e \neq e_1, \ldots, e_{k-1}, \text{ and } \{e_1, \ldots e_{k-1}, e\} \in \mathcal{F}\} .$$

The algorithm stops when no such element exists. We call the set $X_{\mathrm{gr}} = \{e_1, \ldots, e_r\}$ obtained by this algorithm a *greedy solution*, and let X_{opt} denote an optimum solution to our problem. Since we are maximizing, trivially

$$c(X_{\mathrm{gr}}) \geq c(X_{\mathrm{opt}}).$$

Define, for $X \subseteq E$, the *upper rank* of X by

$$r(X) = \max\{|Y| : \ Y \subseteq X, \ Y \in \mathcal{F}\}.$$

We also define the **lower rank** of X by

$$\rho(X) = \min\{|Y| : Y \subseteq X, \ Y \in \mathcal{F}, \ \nexists U \in \mathcal{F} \text{ with } Y \subset U \subseteq X\}.$$

Note that matroids are just those hereditary families with $r = \rho$. In general, define the *rank quotient* of $(E, \mathcal{F})$ by

$$\gamma_{\mathcal{F}} = \min\left\{\frac{\rho(X)}{r(X)} : \ X \subseteq E, \ r(X) > 0\right\}.$$

Note that $\gamma_{\mathcal{F}}$ is just the worst ratio between $c(X_{\mathrm{opt}})$ and $c(X_{\mathrm{gr}})$ for 0-1 valued weight functions c. The following theorem of Jenkyns (1976) and Hausmann and Korte (1978) gives a performance guarantee for the greedy algorithm by showing that 0-1 weightings are the worst case.

Theorem 2.2 *For every hereditary family $(E, \mathcal{F})$ and every weight function $c : E \to \mathbb{Z}_+$, we have*

$$c(X_{\mathrm{gr}}) \geq \gamma_{\mathcal{F}} c(X_{\mathrm{opt}}).$$

∎

Proof. In other words, 0-1 weightings are the worst for the greedy algorithm.

The proof of this theorem depends on the following construction. Order $E = (e_1, e_2, \dots, e_n)$ by decreasing weight:

$$c(e_1) = \dots = c(e_{i_1}) > c(e_{i_1+1}) = \dots = c(e_{i_2}) > \dots$$

and let

$$A_k = \{e_1, \dots, e_{i_k}\}, \qquad \lambda_k = c(e_{i_k}) - c(e_{i_k+1}).$$

Then we can write the weight function c as

$$c = \sum_k \lambda_k \chi^{A_k},$$

where χ^A denote the incidence function of the set A. The key observation is that X_{gr} is also a greedy solution for the weight function χ^{A_k}, and hence

$$\chi^{A_k}(X_{\mathrm{gr}}) \geq \rho(A_k) \geq \gamma r(A_k).$$

Hence

$$c(X_{\mathrm{gr}}) = \sum_k \lambda_k \chi^{A_k}(X_{\mathrm{gr}}) \geq \gamma \sum_k \lambda_k r(A_k).$$

On the other hand,

$$c(X_{\mathrm{opt}}) = \sum_k \lambda_k \chi^{A_k}(X_{\mathrm{opt}}) \leq \sum_k \lambda_k r(A_k).$$

This proves the theorem.

As an application, consider the greedy heuristic for the matching problem. Here G is a graph, $E = E(G)$, and $\mathcal{F}$ consists of all matchings, i.e., sets of edges with no common endnode. We claim that $\gamma \geq 1/2$. In fact, if $X \subseteq E$ and M is a smallest non-extendible matching in X then the $2|M|$ endpoints of the edges in M cover all edges in X, and hence a maximum matching in X cannot have more that $2|M|$ edges.

Thus we obtain that *for every weighting, the greedy algorithm produces a matching with at least half the maximum weight.* (We remark that, as we shall see, the optimum matching can be found in polynomial time by a much more complicated algorithm.)

This result about matchings extends to matchings in hypergraphs. A *hypergraph with rank* r is a pair $(V, \mathcal{H})$, where V is a finite set whozxe elements are called *nodes*, $\mathcal{H} \subseteq 2^V$ and $|X| \le r$ for every $X \in \mathcal{H}$. The members of $\mathcal{H}$ are called edges. A *matching* in a hypergraph is a collection of disjoint edges. Let $\mathcal{F}$ consists of all matchings. For this hereditary set-system, $\gamma \ge 1/r$ (and it can be as small as $1/r$). The proof of these fact is left as an exercise.

As an application, consider the greedy heuristic for the matching problem. Here G is a graph, $E = E(G)$, and $\mathcal{F}$ consists of all matchings, i.e., sets of edges with no common endnode. We claim that $\gamma_{\mathcal{F}} \ge 1/2$. In fact, if $X \subseteq E$ and M is a smallest non-extendible matching in X then the $2|M|$ endpoints of the edges in M cover all edges in X, and hence a maximum matching in X cannot have more that $2|M|$ edges. Thus we obtain that *for every weighting, the greedy algorithm has performance ratio at most* 2. **Greedy blocking sets.** We discuss a greedy heuristic with a somewhat

more involved analysis. Let $(V, \mathcal{H})$ be a hypergraph. A *blocking set* or *cover* of the hypergraph is a subset $S \subseteq V$ that meets (blocks) every member of $\mathcal{H}$. Let S_{opt} denote a blocking set with a minimum number of elements, and define the *covering number* by $\tau(\mathcal{H}) = |S_{\text{opt}}|$. (To compute $\tau(\mathcal{H})$ is NP-hard in general; cf. also chapters 7 and 24.) A *greedy blocking set* S_{gr} is constructed as follows. Choose a vertex v_1 of maximum degree. If $v_1, \ldots, v_k$ have been selected, choose v_{k+1} to block as many members of $\mathcal{H}$ not blocked by $v_1, \ldots, v_k$ as possible. We stop when all members of $\mathcal{H}$ are blocked. Concerning the performance of this algorithm, we have the following bound (Johnson 1974, Stein 1975, Lovász 1976).

Theorem 2.3 *Let $\mathcal{H}$ be a hypergraph with maximum degree Δ. Then for the size of any greedy set-cover,*

$$|S_{\text{gr}}| \le \left(1 + \frac{1}{2} + \ldots + \frac{1}{\Delta}\right) \tau(\mathcal{H}).$$

(The "error factor" on the right hand size is less than $1 + \ln r$. It is easy to see that the ratio $1 + 1/2 + \ldots + 1/\Delta$ cannot be improved.) **Proof.**

Let k_i denote the number of vertices in S_{gr} selected in the phase when we were able to block i new edges at a time. Let $\mathcal{H}_i$ denote the set of edges first blocked in this phase. So $|\mathcal{H}_i| = ik_i$, and

$$|S_{\mathrm{gr}}| = k_\Delta + k_{\Delta-1} + \ldots + k_1.$$

Now consider the optimum set-cover S_{opt}. Every vertex in S_{opt} (in fact, every vertex of $\mathcal{H}$), blocks at most i edges from $\mathcal{H}_i \cup \mathcal{H}_{i-1} \cup \ldots \mathcal{H}_1$, since a vertex blocking more from this set should have been included in the greedy cover before phase i. Hence

$$|S_{\mathrm{opt}}| \geq \frac{1}{i}(ik_i + (i-1)k_{i-1} + \ldots + k_1). \tag{1}$$

Multiplying this inequlity by $1/(i+1)$ for $i = 1, \ldots, \Delta - 1$, and by 1 for $i = \Delta$, and then adding up the resulting inequalities, we obtain that

$$(1 + \frac{1}{2} + \ldots + \frac{1}{\Delta})|S_{\mathrm{opt}}| \leq k_1 + \ldots + k_\Delta = |S_{\mathrm{gr}}|. \blacksquare$$

Note that in this proof, we do not directly compare $|S_{\mathrm{gr}}|$ with $|S_{\mathrm{opt}}|$; the latter is not available. Rather, we use a family (1) of lower bounds on $|S_{\mathrm{opt}}|$. For each i, $(1/i)(\mathcal{H}_1 \cup \ldots \cup \mathcal{H}_i)$ can be viewed as a *fractional matching* (see Chapters 7, 24), and so the theorem can be sharpened by using the fractional matching (or cover) number τ^* on the right hand side. In fact, the fractional cover number τ^* is a relaxation of the cover number, obtained by formulating it as an integer linear program and dropping the integrality constraints. **Greedy travelling salesman tours.** Recall the

travelling salesman problem: given a graph $G = (V, E)$ on $n \geq 3$ nodes, and "distances" $c : E \to \mathbb{Z}_+$, find a Hamilton circuit with minimum length. Usually it is not an essential restriction of generality to assume that G is a complete graph. Moreover, in many important applications, the distance function c satisfies the triangle inequality: $c_{ij} + c_{jk} \leq c_{ik}$ for any three nodes i, j and k. We shall always restrict ourselves to the special case of the complete graph with lengths satisfying the triangle inequality (called the *metric case*). The travelling salesman problem is NP-hard. Linear programming provides a practically quite efficient method to solve it, as we shall see. Here we discuss how a greedy heuristic, called NEAREST INSERTION, works for this problem. We build up a circuit going through more and more nodes. — Start with any node v_1 (viewed as a circuit with one node). — Let T_k be a circuit of length k already constructed. Choose a node v_k not on T_k and a node u_k on T_k such that the distance $c_{u_k v_k}$ is minimal. Delete one of the edges of T_k incident with u_k, and connect its endpoints to v_k, to get a circuit T_{k+1}. — After n steps we get a Hamilton circuit T_{nins}. Another way to describe this heuristic is the following. The tree F formed by the edges $v_k u_k$ is constructed by Prym's algorithm, and so it is a shortest spanning tree of G. We double each edge of it to obtain an Eulerian graph. An Euler tour of this visits every vertex at least once; making shortcuts, we obtain a tour that visits all nodes exactly once.

Theorem 2.4 *T_{nins} is at most twice as long as the optimum tour.*

Proof. Let T_{opt} denote an optimum tour. We make two observations. First, deleting any edge from T_{opt} we get a spanning tree. Hence

$$c(T_{\text{opt}}) \geq c(F). \tag{2}$$

Second, inserting v_k into T_k we increase its length by at most $2c_{u_k v_k}$; adding up these increments, we get

$$c(T_{\text{nins}}) \leq 2c(F). \blacksquare$$

Note that this argument (implicitly) uses a *dual heuristic* also: inequality (2) makes use of the fact that the length of the shortest spanning tree is a lower bound on the length of the shortest tour. The value of this dual heuristic is easily found by the greedy algorithm. In fact, a somewhat better lower bound could be obtained by considering *unicyclic* subgraphs, i.e. those subgraphs containing at most one circuit. The edge-sets of such subgraphs also form a matroid. The bases of this matroid are connected spanning subgraphs containing exactly one circuit; for this chapter, we call such subgraphs *1-trees.* A 1-tree with minimum length can be found easily by the greedy algorithm, and since every tour is a 1-tree, this gives a lower bound on the minimum tour. We'll see later how to further improve this lower bound. There are several other greedy-like heuristics for TSP, such as farthest insertion, cheapest insertion, nearest neighbor, savings etc. Most of them do not have, however, a proven constant performance ratio,although some of them work better in practice than the nearest insert (see Reinolt 1993). The heuristic for the Traveling Salesman Problem with the best known performance ratio is due to Christofides (1976). It uses the shortest spanning tree F, but instead of doubling its edges to make it Eulerian, it adds a matching M with minimum length on the set of nodes with odd degree in F. (Such a matching can be found in polynomial time, see Chapter 3.) It is easy to see that $2c(M) \leq c(T_{\text{opt}})$, and hence the Christofides heuristic has performance ratio of at most 3/2. **Bin packing.** Let $a_1, \ldots, a_n > 0$ be real numbers ("weights"). We would like to partition them into classes ("bins") $B_1, \ldots, B_k$ so that the total weight of every bin is at most 1. We would like to minimize the number k of bins. Let k_{opt} be this minimum. To compute k_{opt} is NP-hard. A trivial lower bound on how well we can do is the total weight $w = \sum_i a_i$. The following simple (greedy) heuristic already gets asymptotically within a factor of 2 to this lower bound. NEXT-FIT HEURISTIC. We process $a_1, a_2, \ldots$ one by one. We put them into one bin until putting the next a_i into the bin would increase its weight over 1. Then we close the bin and start a new one. We denote by k_{NF} the number of bits used by this heuristic.

Theorem 2.5 *$k_{\text{nf}} < 2w + 1$; $k_{\text{nf}} \leq 2k_{\text{opt}}$.*

Proof. Let $k = k_{\text{nf}}$, and denote by w_i the weight put in bin B_i by the NEXT-FIT heuristic ($1 \leq i \leq k$). Then clearly $w_i + w_{i+1} > 1$ (since otherwise the first weight in B_{i+1} should have been put in B_i). If k is even, this implies

$$w = (w_1 + w_2) + \ldots + (w_{k-1} + w_k) \geq k/2,$$

and hence

$$k \leq 2w \leq 2k_{\text{opt}}.$$

If k is odd, then similar calculation implies

$$w = (w_1 + w_2) + \ldots + (w_{k-2} + w_{k-1}) + w_k > \frac{k-1}{2},$$

whence

$$k < 2w + 1 \leq 2k_{\text{opt}} + 1,$$

and hence, $k \leq 2k_{\text{opt}} - 1 < 2k_{\text{opt}}$. ∎

The following better heuristic is still very "greedy". FIRST-FIT HEURISTIC. We process $a_1, a_2, \ldots$ one by one, putting each a_i into the first bin into which it fits. We denote by k_{ff} the number of bins used by this heuristic. The following bound on the performance of First Fit, due to Johnson (****), is substantially more difficult to prove than the previous.

Theorem 2.6 $k_{\text{ff}} \leq 17/10 k_{\text{opt}} + 2$. *There exist lists with arbitrarily large total weight for which* $k_{\text{ff}} = 17/10 k_{\text{opt}}$. ∎

A natural (still "greedy") improvement on the FIRST-FIT heuristic is to preprocess the weights by ordering them decreasingly, and then apply FIRST-FIT. We call this the FIRST-FIT DECREASING heuristic, and denote the number of bins it uses by k_{ffd}. This preprocessing does improve the performance, as shown by the following theorem, due to ****

Theorem 2.7 $k_{\text{ffd}} \leq 11/9 k_{\text{opt}} + 4$. *There exist lists with arbitrarily large total weight for which* $k_{\text{ffd}} \geq 11/9 k_{\text{opt}}$. ∎

There are other greedy-like heuristics for bin-packing, e.g. best fit, whose performance ratio is similar. More involved heuristic algorithms for the bin packing problem use linear programming and achieve an asymptotic performance ratio arbitrarily close to 1; we shall discuss this later. The first two heuristics above are special in the sense that they are *on-line*: each weight is placed in a bin without knowing those that follow, and once a weight is placed in a bin, it is never touched again. On-line heuristics cannot achieve as good a performance ratio as general heuristics; it is easy to show that for every on-line heuristic there exists a sequence of weights (with arbitrarily large total weight) for which it uses at least $(3/2)k_{\text{opt}}$ bins, and this lower bound can be improved to ****. **The knapsack problem.** Given a knapsack with total capacity b, and n objects with weights $a_1, \ldots, a_n$ of value $c_1, \ldots, c_n$, we want to pack as much value into the knapsack as possible. In other words, given positive integers $b, a_1, \ldots a_n, c_1, \ldots c_n$, we want to maximize $\sum_i c_i x_i$ under the constraints

$\sum_i a_i x_i \le b$ and $x_i \in \{0,1\}$. To exclude trivial cases, we may assume that $a_1, \ldots, a_n \le b$. We denote by C_{opt} the optimum value of the objective function. The knapsack problem is NP-hard; we'll see later, however, that one can get arbitrarily close to the optimum in polynomial time. Here we describe a greedy algorithm that has performance ratio at most 2. It is natural that we want to select objects with small weight and large value; so it is natural to put in the knapsack an object with largest "weight density" c_i/a_i. Assume that the objects are labelled so that $c_1/a_1 \ge c_2/a_2 \ge \ldots \ge c_n/a_n$. Then the algorithm consists of selecting $x_1, x_2, \ldots, x_n \in \{0,1\}$ recursively as follows:

$$x_i = \begin{cases} 1, & \text{if } a_i \le b - \sum_{j=1}^{i-1} c_j x_j \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 2.8 *For the solution $x_1, \ldots, x_n$ obtained by the (weight density) greedy algorithm, we have*

$$\sum_{i=1}^{n} c_i x_i > C_{\mathrm{opt}} - \max_i c_i.$$

It follows that comparing the greedy solution with the best of the trivial solutions (select one object), we get a heuristic with performance ratio at most 2. **Proof.** Let x be the solution found by the greedy algorithm, and

y, an optimum solution. If x is not optimal, there must be an index j such that $x_j = 0$ and $y_j = 1$; consider the least such j. Then we have for the greedy solution,

$$\begin{aligned} C_{\mathrm{gr}} &= \sum_{i=1}^{n} c_i x_i \ge \sum_{i=1}^{j} c_i x_i = \sum_{i=1}^{j} c_i y_i + \sum_{i=1}^{j} c_i (x_i - y_i) \qquad (3) \\ &\ge \sum_{i=1}^{j} c_i y_i + \sum_{i=1}^{j} \frac{c_j}{a_j} a_i (x_i - y_i) = \sum_{i=1}^{j} c_i y_i + \frac{c_j}{a_j} \Big(\sum_{i=1}^{j} a_i x_i - \sum_{i=1}^{j} a_i y_i \Big). \end{aligned}$$

Since j was not chosen by the greedy algorithm, we have here

$$\sum_{i=1}^{j} a_i x_i \ge b - a_j$$

Furthermore, the feasibility of y implies that

$$\sum_{i=1}^{j} a_i y_i \le b - \sum_{i=j+1}^{n} a_i y_i.$$

Substituting in (3), we obtain:

$$C_{\text{gr}} \ge \sum_{i=1}^{j} c_i y_i + \frac{c_j}{a_j}\Big(\sum_{i=j+1}^{n} a_i y_i - a_j\Big) \ge \sum_{i=1}^{n} a_i y_i - c_j = C_{\text{opt}} - c_j.$$

∎

3 Matroids

Recall the following general setting for the greedy algorithm from the last section: E is a finite set, $c : E \to \mathbb{Z}_+$, a cost function on E, and $\mathcal{F} \subset 2^E$, a hereditary family of subsets of E, i.e., a set of subsets such that $X \in \mathcal{F}$ and $Y \subseteq X$ implies $Y \in \mathcal{F}$. The goal is to find $\max\{c(X) = \sum_{e \in X} c(e) \mid X \in \mathcal{F}\}$. In this more general setting, We have formulated the greedy algorithm in this general setting: it constructs a maximal set X by finding its elements $e_1, e_2, \ldots$ one by one.

For which families $\mathcal{F}$ does the greedy algorithm find the member of the family with maximum cost?

Definition 3.1 *A* matroid *is a pair* $(S, \mathcal{F})$ *for which the greedy algorithm finds a set* Y *such that* $c(Y) = \max\{c(X) \mid X \in \mathcal{F}\}$.

The "standard" definition of matroids is the following:

Definition 3.2 *A* matroid *is a pair* $(S, \mathcal{F})$ *which satisfies the following three conditions:*

1. $\emptyset \in \mathcal{F}$.

2. $\mathcal{F}$ is hereditary (i.e. closed under taking subsets).

3. $\forall X, Y \in \mathcal{F}, \quad |X| > |Y| \implies \exists x \in X \setminus Y \qquad | \, Y \cup \{x\} \in \mathcal{F}.$

Theorem 3.1 *The two definitions of a matroid are equivalent.*

Proof.

$1 \implies 2:$ We want to show that if the greedy algorithm works, then property 3 is satisfied. Let $X, Y \in \mathcal{F}$ s.t. $|X| > |Y|$. Define a cost function c by

$$c(e) = \begin{cases} 1+\epsilon & \text{if } e \in Y \\ 1 & \text{if } e \in X \setminus Y \\ 0 & \text{otherwise.} \end{cases}$$

The maximum set has cost at least $c(X) = |X| > c(Y)$. The greedy algorithm first takes all the elements of Y and then must continue by taking an element of X.

$2 \implies 1:$ Straightforward generalization of the correctness proof of Kruskal's algorithm.

3.1 Examples

1: For a graph G, $S = E(G)$, $\mathcal{F} = \{E(H) \mid H \subset G, \ H \text{ forest } \}$. To prove that $\mathcal{F}$ satisfies (3), we note that the number of components of a forest Y is $n - |Y|$, thus if $|X| > |Y|$, then X has fewer connected components than Y and must contain an edge connecting two components of Y.

2: Linear matroids. The elements of E are vectors of a given length r over a field, and $\mathcal{F}$ consists of those subsets that are linearly independent. Equivalently, the elements are columns of a given matrix M.

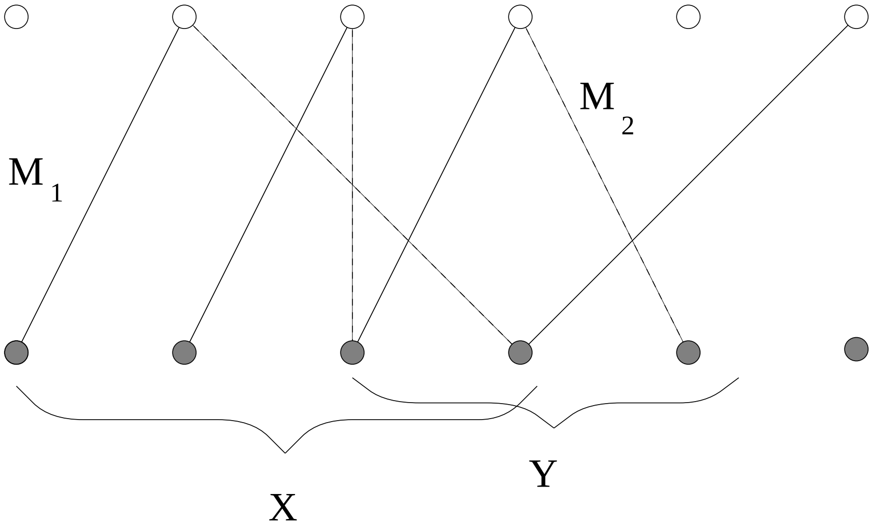

Figure 1: graph of $M_1 \cup M_2$.

3: For a bipartite graph (A, B), on $S = A$, $\mathcal{F} = \{X \subset A \mid$ there is a matching in G which covers $X\}$. The proof requires verifying property 3. Let $X, Y \in \mathcal{F}$, $|X| > |Y|$, M_1 matching of X, and M_2 matching of Y. In the graph with edges $M_1 \cup M_2$, every vertex has degree 0,1 or 2, so the graph consists of isolated points, paths and cycles. Since $|X| > |Y|$, there is at least one odd-length path with an extremity in $X \setminus Y$. We can replace the edges on this path coming from M_2 by the edges coming from M_1 (see figure 3.1), thus getting a matchable set containing Y and an element of $X \setminus Y$.

4: For a connected graph G, $S = E(G)$, $\mathcal{F} = \{X \subseteq E \mid G \setminus X$ is connected $\}$. To prove that $(E, \mathcal{F})$ satisfies (3), consider $G \setminus Y$ (which is connected) and assume (by way of contradiction) that every edge in $X \setminus Y$ is a cut-edge of $G \setminus Y$. Then $G \setminus Y \setminus (X \setminus Y)$ has $|X \setminus Y| + 1 > |Y \setminus X| + 1$ connected components, and so putting back the edges of $Y \setminus X$ cannot make it connected. But this implies that $X \notin \mathcal{F}$.

3.2 Definitions and basic properties

We now give a few basic properties and definitions.

Lemma 3.1 *The maximal elements of $\mathcal{F}$, called the* bases *of the matroid, all have the same cardinality, called* rank. *Given $S' \subset S$, all the maximal elements of $\mathcal{F}$ contained in S' have the same cardinality and are called* bases of S'.

The fact that the bases of S' all have the same size is a direct consequence of (3). This property in fact characterizes matroids: if $(E, \mathcal{F})$ satisfies M_1 and M_2, and for all $S' \subseteq S$, all the maximal subsets of S' that belong to $\mathcal{F}$ have the same cardinality, then $(S, \mathcal{F})$ is a matroid. Property (3) follows by taking $S' = X \cup Y$.

For a general matroid $(S, \mathcal{F})$, any weight function $w : S \to \mathcal{R}_+$ defines a weight on the elements of $\mathcal{F}$, $w(X) = \sum_{e \in X} w(e)$. We are interested in the maximum weight elements of $\mathcal{F}$. In case 3 above, maximum weight elements are trees.

Definition 3.3 *A* greedy basis *with respect to w is defined recursively by $e_k \neq e_1, \ldots, e_{k-1}$, $(e_1, \ldots, e_{k-1}, e_k) \in \mathcal{F}$ and*

$$w(e_k) = \max\{w(e) \mid e \neq e_1, \ldots e_{k-1} \text{ and } (e_1, \ldots, e_{k-1}, e) \in \mathcal{F}\} \ .$$

Theorem 3.1 states that if $(S, \mathcal{F})$ satisfies (1) and (2), then it is a matroid iff $\forall w : S \to \mathcal{F}_+$, the greedy basis is optimal (has maximum weight). To prove the "if" part, let $S' \subseteq S$; we prove that the bases of S' all have the same size. We define the weight function $w(e) = 1$ if $e \in S'$, and $w(e) = 0$ otherwise; then if X_1 and X_2 are two bases of S', they can be extended into bases B_1 and B_2 of S with weight $|X_1|$ and $|X_2|$ respectively. Moreover, both these bases are greedy bases, and hence $|X_1| = |X_2|$.

In the opposite direction, we can say much more about the greedy basis.

Lemma 3.2 *Let $(e_1, \ldots e_r)$ be the greedy basis, and $(f_1, \ldots f_r)$ be any basis where the elements f_i are ordered by $w(f_1) \geq w(f_2) \geq \ldots \geq w(f_r)$. Then $\forall i$, we have $w(f_i) \leq w(e_i)$.*

The proof, easy, works by contradiction; taking the first k such that $w(f_k) > w(e_k)$, we observe that for any $i < k$, $w(f_i) \geq w(f_k) > w(e_k)$. Hence the greedy algorithm choosing e_k implies that $\forall i \leq k$, $(e_1 \ldots e_{k-1}, f_i) \notin \mathcal{F}$, which contradicts M_3 of the matroid definition for $(e_1, \ldots, e_{k-1})$ and $(f_1, \ldots, f_k)$.

The greedy algorithm consists in optimistically taking at each step the best choice; a similar pessimistic algorithm consists in trying to discard at each step the worst choice, i.e. the least weight edge. For example, for the forest matroid above (example 3), a least weight edge can be discarded unless it disconnects the graph. The spanning tree obtained is the *greedy pessimistic* spanning tree. This leads to the concept of *dual* matroids.

Definition 3.4 *Given a matroid* $(S, \mathcal{F})$*, let* $\mathcal{F}^* = \{X \subset S \mid \exists$ *basis* B *of* $\mathcal{F}$ *such that* $B \subset S \setminus X\}$.

Lemma 3.3 *The system* $(S, \mathcal{F}^*)$ *is also a matroid, called the* dual matroid *of* $(S, \mathcal{F})$*. It satisfies*

$$\mathcal{F}^{**} = \mathcal{F} \ .$$

M_1 and M_2 being trivially true, we need to show that $\mathcal{F}^*$ satisfies M_3. Consider X and Y in $\mathcal{F}^*$ such that $|X| > |Y|$. Let B_X be the basis of $\mathcal{F}$ disjoint from X, and B_Y the basis of $\mathcal{F}$ disjoint with Y. If $X \setminus Y \not\subseteq B_Y$ then we can augment Y by any element of $X \setminus (Y \cup B_Y)$. So suppose that $X \setminus Y \subseteq B_Y$. Apply M_3 to $B_Y \setminus X$ and $B_X \setminus Y$. Since $|B_Y \setminus X| = r - |X \setminus Y| < r - |Y \setminus X| \leq |B_X \setminus Y|$, you obtain an element $b \in B_X \setminus Y$ such that $(B_Y \setminus X) \cup \{b\} \in \mathcal{F}$. Apply M_3 to sets $(B_Y \setminus X) \cup \{b\}$ and B_Y to extend the first set until you get a basis B after $|X| - 1$ extensions. $B = B_Y - x \cup b$, where $x \in X$ and $b \notin X \cup Y$. Therefore B is disjoint from $Y \cup x$ and $Y \cup x \in \mathcal{F}^*$. See figure 2.

In the general setting, the worst-case greedy algorithm works by choosing e_i such that $w(e_i) = \min\{w(e) \mid e \neq e_1, \ldots e_{i-1}$, and $S \setminus \{e_1, \ldots, e_{i-1}, e\}$ contains a basis$\}$, which means that $\{e_1, \ldots e_{i-1}, e\} \in \mathcal{F}^*$. Transforming w into $w' = K - w$ with K big enough makes it the greedy algorithm for w' and $\mathcal{F}^*$, hence the following property:

Lemma 3.4 *The worst-case greedy algorithm gives a basis of maximum weight.*

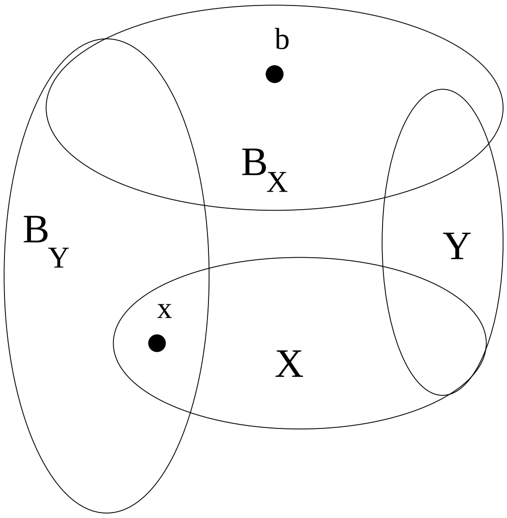

Figure 2: The dual of a matroid is a matroid.

(A good exercise would consist in proving the above lemma directly). How many matroids are there in the world?

Lemma 3.5 *If S has n elements, then there are at least $2^{2^{cn}}$ matroids on S, where c is a positive constant.*

The proof uses a special family of matroids.

Definition 3.5 *If S_i denotes the family of all subsets of S of size i, then a* paving matroid *is of the form*

$$\mathcal{F} = S_0 \cup S_1 \cup \ldots \cup S_{r-1} \cup B \ ,$$

where $B \subset S_r$ is such that

$$X, Y \notin B, \quad |X| = |Y| = r \Longrightarrow |X \cap Y| \leq r - 2 \ .$$

It is easy to see that properties M_1 and M_2 are satisfied by these families. We now prove M_3: the only non trivial case is for $X \in B$, $|Y| = r - 1$, and $Y \not\subset X$, which implies that X has at least two elements x_1 and x_2

which are not in Y. Since $|(Y \cup x_1) \cap (Y \cup x_2)| = r - 1$, one of these two sets has to be in B, hence M_3.

We now want to use these families to show that there are many matroids on S. Let $\overline{B} = S_r \setminus B$. Assuming that n is divisible by 4, take $r = n/2$. $\mathcal{F}$ is defined by $\overline{B}$, and if $\overline{B}$ defines a paving matroid, then any subset of $\overline{B}$ also does. Match the n elements into $n/2$ pairs. Define $\overline{B} = \{X \mid X$ is a union of $n/4$ pairs$\}$. Then it defines a paving matroid. By taking subsets, we get

$$2^{\binom{n/2}{n/4}}$$

paving matroids of S, which proves the lemma.

These paving matroids can be used to obtain various counterexamples. For example, the paving matroid defined by $S = \{1, 1', 2, 2', 3, 3', 4, 4'\}$ and $\overline{B} = \{\{1, 1', 2, 2'\},\ \{1, 1', 3, 3'\},\ \{1, 1', 4, 4'\},\ \{2, 2' 3, 3'\}, \{3, 3', 4, 4'\}\}$ (the so-called *Vamos matroid*), is not a linear matroid over any field.

Thus there can not be any compact encoding of matroids. A matroid is given by an oracle which answers the question "$X \in \mathcal{F}$?"; the worst-case greedy algorithm seems to require an $\mathcal{F}^*$-oracle, but can actually use an $\mathcal{F}$ oracle (with which ranks can be determined).

Historically, these notions were first introduced in connection with spanning trees of planar graphs. Given a planar graph G, we can define the planar dual graph G^* (see figure 3). The families of their forests are matroids. There is a bijection between $E(G)$ and $E(G^*)$, and through it a cycle of G is mapped to a separator of G^*, and conversely. The complement of a spanning tree of G is mapped to a spanning tree of G^*. Thus the two matroids are dual of one another.

In general, a non-planar graph has a matroid but the dual of that matroid does not necessarily come from a graph (in fact this property characterizes planar graphs).

3.3 Rank function.

For a matroid, the upper rank and the lower rank function are equal (e.g. by Theorem 2.2). The rank function has many nice properties.

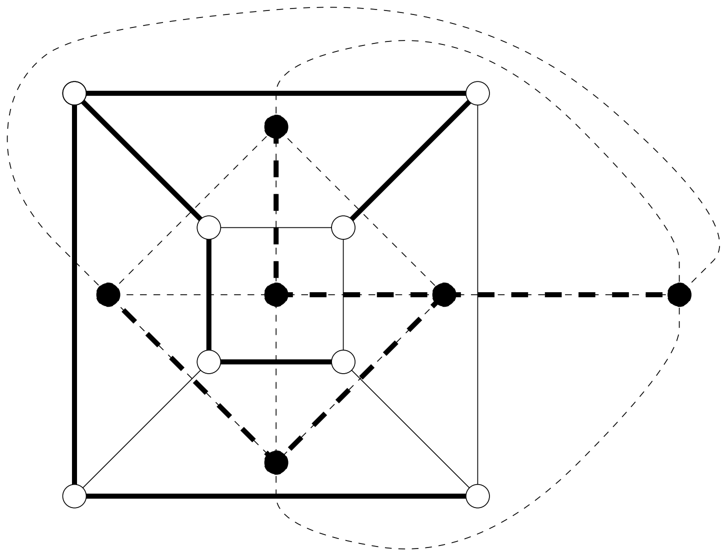

Figure 3: A planar graph and its dual.

Lemma 3.6 *The rank function satisfies the following properties: For all* $X, Y \subseteq S$*,*

$$r(\emptyset) = 0.$$

$$Y \subset X \Longrightarrow r(Y) \leq r(X).$$

$$\forall X,\ r(X) \leq |X|.$$

$$\forall X, Y,\ r(X \cup Y) + r(X \cap Y) \leq r(X) + r(Y) \text{ (i.e. } r \text{ is submodular).}$$

The first three properties are trivial; to prove the fourth one, we take A basis of $X \cap Y$, extend A to a basis B of X, extend B to a basis C of $X \cup Y$. The ranks of X, $X \cap Y$ and $X \cup Y$ are equal to $|B|$, $|A|$ and $|C|$ respectively. Since $C \cap Y \in \mathcal{F}$, we have $r(Y) \geq |C \cap Y|$. Thus it is sufficient to show that

$$|C \cap Y| + |B| - |C| - |A| \geq 0\ .$$

Any element of A is counted in all four terms, which cancel; any element of $B \setminus A$ is counted in C and B, which cancel; any element of $C \setminus B$ is counted in C and $C \cap Y$, since the extension of B into C was done by adding elements of $Y \setminus X$ only. Thus everything cancels, which proves the property.

Theorem 3.2 *These properties characterize matroid rank functions.*

Given a function defined on the subsets of S and which satisfies the properties of the lemma, the only matroid for which it can be the rank function is the one defined by

$$X \in \mathcal{F} \Longleftrightarrow |X| = r(X) .$$

Proof that $(S, \mathcal{F})$ is a matroid: The empyset obviously is in $\mathcal{F}$. If $Y \subset X \in \mathcal{F}$, then we have $r(X) = r(Y) + r(X \setminus Y)$, hence

$$\begin{aligned} |X| &\leq |X \setminus Y| + r(Y) \\ &\leq |X \setminus Y| + |Y| \\ &\leq |X| \end{aligned}$$

and hence all the inequalities must be equalities. For proving the fourth property, we consider $X, Y \in \mathcal{F}$ such that $|X| < |Y|$, and add the elements of $Y \setminus X$ to X one by one until the rank increases. Thus

$$\begin{aligned} r(X \cup a_1 \cup \ldots \cup a_{i-1}) &= r(X) = |X| \\ r(X \cup A_1 \cup \ldots \cup a_i) &= |X| + 1 \end{aligned}$$

and applying the submodularity property to $X \cup a_i$ and $X \cup a_1 \cup \ldots \cup a_{i-1}$ shows that $X \cup a_i \in \mathcal{F}$.

We now introduce the notion of closure of a set. Given a subset X of S, we consider all the sets Y containing X and such that $r(Y) = r(X)$. We call their union $\overline{X}$. A direct application of the submodularity of the rank function shows that $r(\overline{X}) = r(X)$.

Definition 3.6 $\overline{X}$ *is called the* closure *of* X*.*

For example, in a linear matroid the rank is the size of a maximal subset of linearly independent vectors of X, and the closure of X is the linear span of the vectors of X.

In the case of a graph and the matroid of the forests of edges, given X subgraph of the graph, $\overline{X}$ is the maximal subgraph which has the same connected components as X.

Lemma 3.7 *The closure operation satisfies the following properties: For all $X, Y \subseteq S$ and $a, b \in S$,*

$$X \subset \overline{X}.$$

$$\overline{\overline{X}} = \overline{X}.$$

$$X \subset Y \Longrightarrow \overline{X} \subset \overline{Y}.$$

$$a \notin \overline{X} \text{ and } a \in \overline{X \cup b} \Longrightarrow b \in \overline{X \cup a}.$$

These properties caracterize matroid closure.

The first two properties are obvious, the next two are consequences of the submodularity of the rank. Given a function on the subsets of S which satisfies these four properties, we define $\mathcal{F}$ by

$$X \in \mathcal{F} \Longleftrightarrow \forall a \in X, \quad a \notin \overline{X \setminus a} \ .$$

Proof that $(S, \mathcal{F})$ is a matroid: exercise.

Definition 3.7 *X is called a* closed set *or a* flat *if $\overline{X} = X$.*

Clearly for every Y, $\overline{Y}$ is closed; the intersection of two closed sets is a closed set. Thus if we know the family of all closed sets, we can define the closure of a set to be the intersection of all closed sets containing it, i.e. the smallest closed set containing it.

Clearly the closure of a basis is S.

Definition 3.8 *A closed set X such that $r(X) = r(S) - 1$ is called a* hyperplane.
A minimal non-independent set is called a circuit.
A complement of a hyperplane is called a cocircuit.

Thus the terminology for matroids reflects the fact that they are an abstraction of independent vectors in a field on the one hand, forests of edges in a graph on the other hand. A cocircuit in the graph theory definition is a minimal set of edges whose deletion increases the number of connected components of the graph. Thus its complement does not contain a spanning tree, but adding any edge of the cocircuit increases the rank to $r(S)$: in other words, the complement of a cocircuit is a hyperplane, which justifies the definition.

Note that a cocircuit can also be defined equivalently as a minimal set which meets all the bases of the matroid, i.e. a minimal set not contained in any basis of the dual. Thus circuits of $\mathcal{F}$ correspond to cocircuits of $\mathcal{F}^*$.

We are now going to define the matroid analog of edge-deletion and contraction in a graph, and of truncation in a vector space.

Definition 3.9 *Let* $(S, \mathcal{F})$ *be a matroid, and* $S' \subset S$. *Then the* deletion *of* $S \setminus S'$ *from the matroid induces the matroid* $(S', \mathcal{F}|_{S'})$, *where* $\mathcal{F}|_{S'} = \{X \in \mathcal{F} \mid X \subset S'\}$.

Let $x \in S$. *Then the* contraction *of* x *induces the matroid* $(S \setminus x, \mathcal{F}/x)$, *where*

$$\mathcal{F}/x = \begin{cases} \{Y \mid Y \cup \{x\} \in \mathcal{F}\} & \textit{if } r(\{x\}) = 1 \\ \mathcal{F} & \textit{otherwise.} \end{cases}$$

Given a subset U *of* S, *the* contraction of U *produces the matroid* $\mathcal{F}/U = \{Y \mid Y \cup \{ \text{ basis of } U\} \in \mathcal{F}\}$ *(note that this is independent of the basis chosen).*

A truncation *of* $(S, \mathcal{F})$ *produces a matroid of the form* $(S, \mathcal{F}^{(k)})$, *where* $\mathcal{F}^{(k)} = \{X \in \mathcal{F} \mid \ |X| \leq k\}$.

We note that deletions and contractions are closely related through passage to the dual: in fact, in the example of the forest matroid for a planar graph, deleting edge e from the graph amounts to merging two regions, i.e. to contracting the corresponding edge in the dual graph (see figure 4). (A side note about planar graphs matroids: looking at the rank of a pair of dual matroids, it can be seen that Euler's formula $(v-1)+(f-1) = e$ is a corollary of the fact that the dual of a matroid is a matroid).

Using the notation

$$\mathcal{F} \setminus U = \mathcal{F}|(S - U) \ ,$$

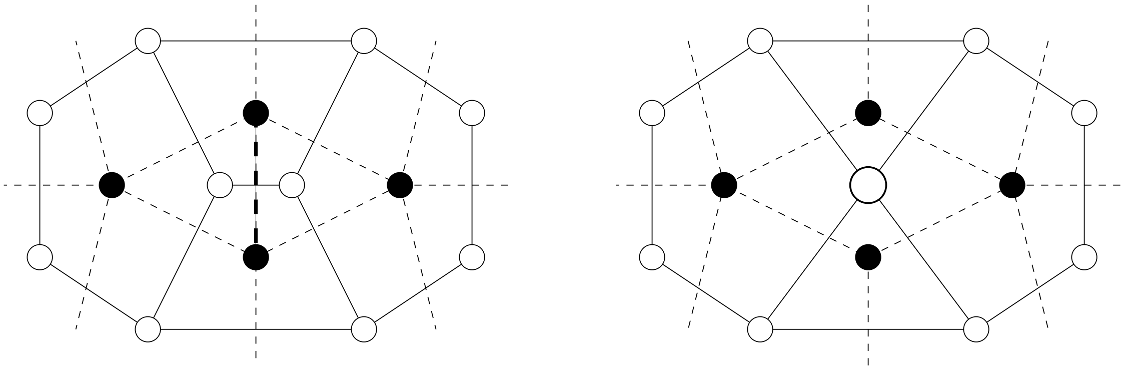

Figure 4: Deletions and contractions are dual operations.

we can state this duality of deletions and contractions in the following way:

Lemma 3.8

$$\mathcal{F} \setminus U = (\mathcal{F}^*/U)^* \ .$$

Definition 3.10 *A* minor *of a matroid* $(S, \mathcal{F})$ *with respect to* X *and* Y *is the matroid* $(S \setminus (X \cup Y), (\mathcal{F}/X) \setminus Y)$.

4 Local improvement

The greedy algorithm and its problem specific versions belong to a class of algorithms sometimes called (one-pass) *construction heuristics.* Former decisions are not reconsidered. The purpose of these heuristics is to find a reasonable solution very fast. But, of course, a locally good choice may lead to a globally poor solution.

Exchange heuristics for the travelling salesman problem. Such unpleasant behaviour is often visually apparent. For instance, if one plots the travelling salesman tour constructed by the *nearest neighbour heuristic*

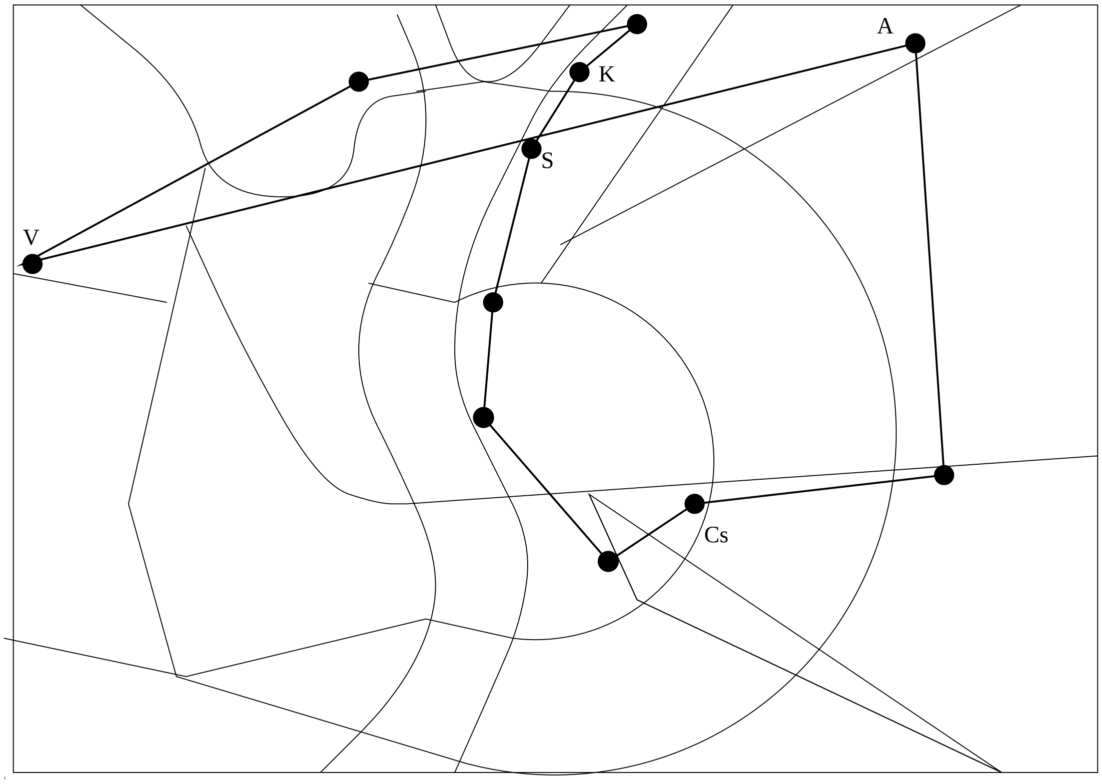

Figure 5: A nearest neighbour tour

(visit some arbitrarily chosen node; from the last node visited go to the closest not yet visited node; if all nodes have been visited return to the first node), one can see that, initially, short connections are chosen but that, in the end, very long steps have to be made to connect the "forgotten" nodes. Figure refnntour shows the nearest neighbour tour of the 11-"city" problem of visiting 10 popular pubs in Budapest, the construction started with node Cs (nearest to the University. This picture obviously calls for a "repair" of the solution.

There is an apparent idea. We remove the edge between A and V, and the edge between S and K and reconnect the remaining two paths by adding the edges between A and K and S and V, respectively, to obtain a new tour. If we perform this exchange iteratively and systematically we obtain a successful improvement technique called *2-opt*.

The *2-opt heuristic* starts with some tour T, for instance a random tour or a tour obtained by a construction heuristic. Then it checks for all pairs of nonadjacent edges uv, xy of T, whether $S := T \setminus \{uv, xy\} \cup \{ux, vy\}$ is shorter than T. If so, T is replaced by S and the exchange tests are repeated. Otherwise the heuristic stops.

There is an obvious generalization of this method: instead of removing two edges, we delete r nonadjacent edges from T. Then we enumerate all possible ways of adding r other edges such that these r edges together with the remaining r paths form a tour. If the shortest of these tours is shorter than the present tour T we replace T by this shorter tour and repeat. This method is called the *r-opt heuristic.*

These heuristics are prototypical *local improvement techniques* that, in general, work as follows. We have some feasible solution of the given combinatorial optimization problem. Then we do some little operation on this solution by removing some elements and adding other elements to obtain one or several new solutions. If one of the new solutions is better than the present one we replace the present one by the new best solution and repeat.

The basic ingredient of such improvement or exchange heuristics is a rule that describes the possible manipulations that are allowed. This rule implicitly defines, for every feasible solution, a set of other feasible solutions that can be obtained by such a manipulation. Using this interpretation, we can define a digraph D whose vertex set is the set Ω of feasible solutions of our combinatorial optimization problem (this is typically exponentially large) and where an arc (S, T) is present if T can be obtained from S by the manipulation rule.

Now, local improvement heuristics can be viewed as algorithms that start at some node of this digraph and search, for a current node, the successors of the node. If they find a better successor they go to this successor and repeat. The term *local search algorithms* that is also used for these techniques derives from this interpretation.

There are many important algorithms that fit in this scheme: the simplex method (see Chapter 19), basis reduction algorithms (Chapter 20) etc. The simplex method is somewhat special in the sense that it only gets stuck when we have reached the optimum. Usually, local improvement algorithms may run into *local optima*, i.e., solutions from which the given local manipulations do not lead to any better solution.

The Max-Cut Problem. Let us look at another example. Suppose we are given a graph $G = (V, E)$ and we want to find a cut at $\nabla(W), W \subseteq V$, of maximum size. (This is *the cardinality max-cut problem* for G.) The problem has an obvious weighted version, where each edge has a (say, rational) weight, and we want to find a cut with maximum weight.

We start with an arbitrary subset $W \subseteq V$. We check whether W (or $V \backslash W$) contains a node w such that less than half of its neighbours are in $V \backslash W$ (or W). If such a node exists we move it from W to $V \backslash W$ (or from $V \backslash W$ to W). Otherwise we stop.

Termination of this *single exchange heuristic* in $O(n^2)$ time is guaranteed, since the size of the cut increases in every step. The cut produced by this procedure obviously has a size that is at least one half of the maximum cardinality of a cut in G, and in fact, it can be as bad as this, as the complete bipartite graph shows.

The single exchange heuristic has an obvious weighted version: we push a node w to the other side if the sum of the weights of edges linking w to nodes on the other side is smaller than the sum of the weights of the edges linking w to nodes on side of w. This version, of course, also terminates in finite time, but the number of steps may be exponential (Haken and Luby 1988) even for 4-regular graphs. On the other hand, Poljak (1993) proved that the single exchange heuristic terminates in polynomial time for cubic graphs (We'll return to this later.)

For a typical combinatorial optimization problem, it is easy to find many local manipulation techniques. For instance, for the max-cut problem we could try to move several nodes from one side to the other or exchange nodes between sides, for the TSP we could perform node exchanges instead of edge exchanges, we could exchange whole sections of a tour, and we could combine these techniques, or we could vary the number of edges and/or nodes that we exchange based on some criteria. In fact, most people working on the TSP view the well known heuristic of Lin and Kernighan (1973) and its variants as the best local improvement heuristic for the TSP known to date (based on the practical performance of heuristics on large numbers of TSP test instances). This heuristic is a dynamic version of the *r-opt* heuristic with varying r.

The last statement may suggest that improvement heuristics are the way to solve (large scale) TSP instances approximately. However, there are some practical and theoretical difficulties with respect to running times, traps and worst-case behaviour.

Running time. A straightforward implementation of the *r-opt* heuristic, for example, is hopelessly slow. It takes $\binom{n}{r}$ tests to check whether a tour can be improved by an r-exchange. Even for $r = 3$, the heuristic runs almost forever for medium size problems on a few thousand nodes. To make this approach practical, a number of modifications limiting the exchange s considered are necessary. They are usually based on insights about the probability of success of certain exchanges, or on knowledge about special structures (an instance might be geometrical, e.g., given by points in the plane and a distance function). Well-designed fast data structures play an important role. The issue of speeding up TSP heuristics is treated in depth, e.g., in Reinelt (1993), Johnson (1990) and Bentley (1990). With these techniques TSP instances of up to a million cities have been solved approximately. Such observations apply to many other combinatorial optimization problems analogously.

It may sound strange, but for many exchange heuristics there is no proof that these heuristics terminate in polynomial time. This is the case even with such a basic and classical algorithm as the simplex method (at least for some pivoting rules)! Although we could prove an $O(n^2)$ running time for the cardinality version of the max-cut heuristic mentioned before, its weighted version is not polynomial, as mentioned above.

A single pass through the loop of r-opt takes $O(n^2)$ time but it is not clear how to bound the number of tours that have to be processed before the algorithm terminates with an *r-opt 2tour*, i.e., a tour that cannot be improved by an r-exchange. Computational experience, however, shows that exchange heuristics usually do not have to inspect too many tours until a “local optimum” is found.

Quality of the approximation. Although the solution quality of local improvement heuristics is often quite good, these heuristics may run into traps, for instance *r-opt* tours, whose value is not even close to the optimum. It is, in general, rather difficult to prove worst-case bounds on the quality of exchange heuristics. For an example where a performance ratio is established, see the basis reduction algorithm in Chapter 19.

It is probably fair to say that, for the solution of combinatorial optimization problems appearing in practice, fast construction heuristics combined with local improvement techniques particularly designed for the special structures of the application are the real workhorses of combinatorial optimization. That is why this machinery receives so much attention in the literature and why new little tricks or clever combinations of old tricks are discussed intensively. Better solution qualities or faster solution times may result in significant savings, in particular for complicated problems of large scale.

Aiming at local optima. There are several examples when we are only interested in finding a local optimum: it is the structure of the local optimum, and not the value of the objective function, that concerns us. A theoretical framework for such "polynomial local search problems" was developed by Johnson, Papadimitriou and Yannakakis (1988). analogously to NP, this class also has complete (hardest) problems; the weighted local max-cut problem is one of these (Schäffer and Yannakakis 1991).

Consider an optimum solution W of the max-cut problem for a graph $G = (V, E)$. Clearly, every node is connected to at least as many points on the opposite side of the cut than on its own side. If we only want a cut with this property, any locally optimal cut (with respect ot the single exchange heuristic) would do.

Assume now that we want to solve the following more general problem: given two functions $f, g : V \to \mathbb{Z}_+$ such that $f(v) + g(v) = d_G(v) - 1$ for all $v \in V$, find a subset $W \subseteq V$ such that for each node v, the number of nodes adjacent to v on its own side of the cut is at most

$$\begin{cases} f(v), & \text{if } v \in W \\ g(v), & \text{if } v \in V \setminus W. \end{cases}$$

It is not difficult to guess an objective function (over cuts) for which such cuts are exactly local optima:

$$\phi(W) = |\nabla(W)| + f(W) + g(V \setminus W).$$

Once this is found, it follows that a cut with the desired property exists, and also that it can be found in polynomial time by local improvement.

A much more difficult, but in principle similar, application of this idea is the proof of Szemerédi's Regularity Lemma (see Chapter 23). Here again a tricky (quadratic) objective function is set up, which is locally improved until a locally almost optimal solution is found; the structure of such a solution is what is needed in the numerous applications of the Regularity Lemma (see also Alon, Duke, Lefmann, Rödl and Yuster (1992) for the algorithmic aspects of this procedure).

A beautiful example of turning a structural question into an optimization problem is Tutte's proof (1963) of the fact that every 3-connected planar graph has a planar embedding with straight edges and convex faces. He considers the edges as rubber bands, fixes the vertices of one face at the vertices of a convex polygon, and lets the remaining vertices find their equilibrium. This means minimizing a certain quadratic objective function (the energy), and the optimality criteria can be used to prove that this equilibrium state defines a planar embedding with the right properties. The algorithm can be used to actually compute nice embeddings of planar graphs. A similar method for connectivity testing was given by Linial, Lovász and A. Wigderson (1988).

Randomized exchange. A very helpful idea to overcome some of these difficulties is to randomize. In a randomized version of local search, a random neighbor of the current feasible solution is selected. If this improves the objective function, the current feasible solution is replaced by this neighbor. If not, it may still be replaced, but only with some probability less than 1, depending on how much worse the objective value at the neighbor is. This relaxation of the strict descend rule may help to jump out of traps and eventually reach a significantly better solution.

A more general way of looking at this method is to consider it as generating a random element in Ω, from some given probability distribution Q. Let $f : \Omega \to \mathbb{R}_+$ be the objective function; then maximizing f over Ω is just the extreme case when we want to generate a random element from a distribution concentrated on the set of optimum solutions. If, instead, we generate a random point w from the distibution Q in which $Q(v)$ is proportional to $\exp(f(v)/T)$, where T is a very small positive number, then with large probability w will maximize f. In fact, a randomized algorithm that finds a solution that is (nearly) optimal with large probability is equivalent to a procedure of generating a random element from a distribution that is heavily concentrated on the (nearly) optimal solutions.

To generate a random element from a distribution over a (large and complicated) set is of course a much more general question, and is a major ingredient in various algorithms for enumeration, integration, volume computation, simulation, statistical sampling, etc. (see Jerrum, Vazirani and Vazirani (1983), Dyer and Frieze (1990), Sinclair and Jerrum (1988), Dyer, Frieze and Kannan (1989), Lovász and Simonovits (1992) for some of the applications with combinatorial flavor). An efficient general technique here is *random walks* or *Markov chains.* Let $G = (\Omega, E)$ be a connected graph on Ω, and assume, for simplicity of presentation, that G is non-bipartite and d-regular. If we start a random walk on G and follow it long enough, then the current point will be almost uniformly distributed over Ω. How many steps does "long enough" mean depends on the spectrum, or in combinatorial terms on global connectivity properties called expansion rate or conductance, of the graph (see also Chapter 31).

The Metropolis filter. In optimization, we are interested in very non-uniform, rather than uniform, distributions. Fortunately, there is an elegant way, called the *Metropolis filter* (Metropolis, Rosenbluth, Rosenbluth, Teller and Teller 1953), to modify the random walk, so that it gives any arbitrary prescribed probability distribution. Let $F : \Omega \rightarrow \mathbb{R}_+$. Assume that we are at node v. We choose a random neighbor u. If $F(u) \geq F(v)$ then we move to u; else, we flip a biased coin and move to u only with probability $F(u)/F(v)$, and stay at v with probability $1 - F(u)/F(v)$.

Let Q_F denote the probability distribution on Ω defined by the property that $Q_F(v)$ is proportional to $F(v)$. The miraculous property of the Metropolis filter is the following:

Theorem 4.1 *The stationary distribution of the Metropolis-filtered random walk is* Q_F.

So choosing $F(v) = \exp(f(v)/T)$, we get a randomized optimization algorithm.

Unfortunately, the issue of how long one has to walk gets rather messy. The techniques to estimate the conductance of a Metropolis-filtered walk are not general enough, although Applegate and Kannan (1991) have been able to apply this technique to volume computation.

Simulated annealing. Coming back to optimization, let us follow Kirckpatrick, Gelatt and Vecchi (1983), and call the elements of Ω *states* (of some physical system), $1/F(v)$ the *energy* of the state, and T the *temperature*. In this language, we want to find a state with minimum (or almost minimum) energy. A random walk means letting the system get into a stationary state at the given temperature.

The main, and not quite understood, issue is the choice of the temperature. If we choose T large, the quality of the solution is poor, i.e. the probability that it is close to being optimal is small. If we choose T small, then there will be barriers of very small probability (or, equivalently, with very large energy) between local optima, and it will take extremely long to get away from a local, but not global optimum.

The technique of *simulated annealing* suggests to start with the temperature T sufficiently large, so that the random walk with this parameter mixes fast. Then we decrease T gradually. In each phase, the random walk starts from a distribution which is already close to the limiting distribution, so there is hope that the walk will mix fast. (A similar trick works quite well in integration and volume computation; see [LS].) Theoretical and practical experiments have revealed that it matters a lot how long we walk in a given phase (*cooling schedule*).

There are many empirical studies with this method; see Johnson, Aragon, McGeoch and Schevon (1989, 1991) or Johnson (1990). There are also some general estimates on its performance (Holley and Stroock 1988, Holley, Kusuoka and Stroock 1989). Examples of problems, in particular of the matching problem, are known where simulated annealing performs badly (Sasaki and Hajek 1988, Sasaki 1991), and some positive results in the case of the matching problem are also known (Jerrum and Sinclair 1989). The conclusion that can be drawn at the moment is that simulated annealing is a potentially valuable tool (if one can find good cooling schedules and other parameters), but it is in no ways a panacea as was claimed in some papers pioneering this topic.

Approaches named taboo search, threshold accept, evolution and genetic algorithms and others are further variants and enhancements of randomized local search. The above judgement of simulated annealing applies to them as well. See Johnson (1990).

5 Enumeration and branch-and-bound

There is a number of interesting combinatorial optimization problems for which beautiful polynomial time algorithms exist. We will explain some of them in subsequent sections. We now address the issue of finding an optimum solution for an NP-hard problem. In the previous two sections we have outlined heuristics that produce some feasible and hopefully good solution. Such a solution may even be optimal right away. But how does one verify that?

The basic trouble with integer programming and combinatorial optimization is the nonexistence of a sensible duality theory. The duality theorem of linear programming (see Chapter 30), for instance, can be used to prove that some given feasible solution is optimal. In the (rare) cases where duality theorems in integral solutions like the max-flow min-cut theorem (see Chapter 2 or 30) or the Lucchesi-Younger Theorem (see Chapter 2) exist, one can usually derive a polynomial time solution algorithm. For NP-hard problems one should not expect to find such theorems. Unfortunately, there is nothing better known than replacing such a theory by brute force.

Trivial running time estimates reveal that the obvious idea of simply enumerating the finitely many solutions of a combinatorial optimization problem is completely impractical in general. For instance, computing the length of all (1/2)15! ($\sim 0.6 * 10^{12}$) tours of the 16 city TSP instance Ulysses 16 takes about 39 hours on a 27 MIPS HP workstation. Even a terraflop computer will be unable to enumerate all solutions of a ridiculously small 30 city problem within its lifetime.

Unless P=NP, there is no hope that we will be able to design algorithms for NP-hard problems that are asymptotically much better than enumeration. However, we can try to bring problem instances of reasonable sizes (appearing in practice, say) into the realm of practical computability by enhancing enumeration with a few helpful ideas.

Branch-and-bound. The idea of the *branch-and-bound* approach is to compute tight upper and lower bounds on the optimum value in order to significantly reduce the number of enumerative steps. To be more specific, let us assume that we have an instance of a minimization problem. Let Ω

be its set of feasible solutions.

To implement branch-and-bound, we need a dual heuristic (relaxation), i.e., an efficiently computable lower bound on the optimum value. This dual heuristic will also be called for certain subproblems.

We first run some construction and improvement heuristics to obtain a good feasible solution, say T, with value $c(T)$, which is an upper bound for the optimum value c_{opt}.

Now we resort to enumeration. We split the problem into two (or more) subproblems. Recursively solving these subproblems would mean straightforward enumeration. We can gain by maintaining the best solution found so far and computing, whenever we have a subproblem, a lower bound for the optimum value of this subproblem, using the dual heuristic. If this value is larger than the value of the best solution found so far, we don't have to solve this subproblem.

To be more specific, let us discuss a bit the two main ingredients, *branching* and *bounding.*

We assume that the splitting into subproblems (the *branching*) is such that the set Ω of feasible solution is partitioned into the sets Ω' and Ω'' of feasible solutions of the subproblems. We also assume that the subproblems are of the same type (e.g., the dual heuristic applies to them). It is also important that branching step requires little bookkeeping and is computationally cheap. If Ω consists of subsets of a set S, then a typical split is to choose an element $e \in S$ and to set

$$\Omega' := \{I \in \Omega \mid e \in I\}, \Omega'' := \{I \in \Omega \mid e \notin I\}.$$

The *bounding* is usually given by a *relaxation*: by problem specific investigations we introduce a new problem, whose set of feasible solutions is Γ, say, such that every solution of $\Omega \subseteq \Gamma$ and the objective function for Ω extends to Γ. Suppose X minimizes the objective function over Γ, then the value $c(X)$ provides a lower bound for c_{opt} since all elements of Ω participated in the minimization process. If X is, in fact, an element of Ω we clearly have found an optimum solution of Ω. (If $X \notin \Omega$, we may still be able to make use of it by applying a construction and/or improvement heuristic that starts with X and ends with a solution Y, say, in Ω. If $c(Y) < c(T)$ we set $T := Y$ to keep track of our current best solution.)

To give some examples, useful relaxations for the symmetric TSP are perfect 2-matchings (unions of disjoint circuits that cover all nodes) or 1-trees (subgraphs with at most one circuit). For the assymmetric TSP a standard relaxation is obtained by considering unions of directed circuits that cover all nodes (which can be easily reduced to a bipartite perfect matching problem), or the r-arborescence problem.

A particularly powerful method are LP-relaxations. (We'll discuss them in detail later.) There are some general methods to improve relaxations; one technique is called Lagrangian relaxation and will be discussed later.

Returning to the algorithm, we maintain a list of unsolved subproblems, and a solution T that is the current best. In the general step, we choose an unsolved subproblem, say Ω^i, from the list and remove it. We optimize the objective function over the relaxation Γ^i of Ω^i. Let X be an optimum solution. There are several possibilities.

- First, X may be feasible for Ω^i. In this case we have found an optimum over Ω^i and can completely eliminate all elements of Ω^i from the enumeration process. One often says that this branch is *fathomed.* If $c(X) < c(T)$ we reset $T := X$ also.
- Second, if $c(X) \geq c(T)$ then no solution in Γ^i and hence no solution in Ω^i has a value that is smaller than the current champion. Hence this branch is also fathomed and we can eliminate all solutions in Ω^i.
- Third, if $c(X) < c(T)$ (and X is not feasible for Ω^i) we have done the computation in vain. (We may still try to make use of X to obtain a solution better than T as above.) We split Ω^i into two or more pieces, and put these on our list of unsolved subproblems.

The branch-and-bound method terminates when the list of subproblems is empty. The iteratively updated solution T is the optimum solution. Termination is, of course, guaranteed if the set Ω of feasible solutions is finite.

Although the global procedure is mathematically trivial, it is a considerable piece of work to make it computationally effective. The efficiency mainly depends on the quality of the lower bound used. Most of the mathematics that is developed for the solution of hard problems is concerned with the invention of better and better relaxations, with their structural properties and with fast algorithms for their solution.

6 Dynamic Programming

The origin of dynamic programming is the modelling of discrete-time sequential decision processes. The process starts at a given initial state. At any time of the process we are in some state and there is a set of states that are reachable from the present state. We have to choose one of these. Every state has a value and our objective is to maximize the value of the terminating state. Virtually any optimization problem can be modeled by a dynamic program. There is a recursive solution for dynamic programs which, however, is not efficient in general. But the dynamic programming model and this recursion can be used to design fast algorithms in cases where the number of states can be controlled. We illustrate this by means of a few examples.

6.1 Dijkstra's algorithm

We begin with a well-known, very simple algorithm (breadth first search) which illustrates the key idea of dynamic programming. Consider the problem of finding the shortest path in a graph from a distinguished vertex s (the source) to a distinguished vertex t (the sink). The solution of this problem is very simple once we embed the problem in the larger family of problems which ask for the shortest path from the source s to *every* other vertex of the graph. Proceed recursively: label the neighbors of the source s with a 1. Label with k those unlabelled vertices which are neighbors of vertices of label $k-1$. Then the label of each vertex is its distance from s.

This idea reappears in the more general problem of finding a minimum weight path from s to t in a graph with weighted edges. (The weight c_{ij} on the edge $\{i, j\}$ may be interpreted as the length of the edge.) Again we embed this problem in the larger family of problems which asks for a shortest (minimum weight) path from s to every other vertex of the graph.

Dijkstra's algorithm recognizes that a path between two vertices may be of minimum weight even if other paths have fewer edges. So if we begin at the source, among all neighbors j of s, find one such that the edge sj has minimum weight c_{sj}. We can confidently assert that the shortest path in the graph from s to this neighbor has length c_{sj}, but we are not sure about the other neighbors of s. So label this one neighbor c_{sj}. You may think of s as labelled with a 0. In the course of the algorithm we maintain a set T of vertices, each of which we have confidently labelled with the minimum weight of a path from s to it. Call this label $d(s, j)$, the distance from s to the vertex j. These minimum weight paths pass only through vertices already in T. At each step of the algorithm, we consider vertices on the frontier of T (vertices not in T which are neighbors of vertices in T). Consider edges ij between vertices i in T and vertices j in this frontier. An upper bound for the minimum weight of a path from s to j is the smallest of the numbers $d(s, i) + c_{ij}$ where i is in T. Find $i \in T$ and j which minimize $d(s, i) + c_{ij}$. We can confidently label such a vertex j with $d(s, j) = d(s, i) + c_{ij}$ and add j to T for the following reason: Any path from s to j must leave the set T at a vertex i' then pass through a vertex j' of the frontier. Hence this competing path from s to j has length at least $d(s, i') + c_{i'j'}$. By our choice of i and j, the path from s to i in T then immediately to j has length no greater $d(s, i') + c_{i'j'}$ (figure 6).

The subset-sum problem. Given positive integers $a_1, a_2, \ldots, a_n, b$, decide whether there exist indices $1 \leq i_1 < i_2 < \ldots < i_k \leq n$ for some k such that $a_{i_1} + \ldots + a_{i_k} = b$.

This problem is NP-complete; however, there is a pseudopolynomial algorithm to solve it.

First, consider an obvious algorithm using enumeration. Clearly the subset-sum problem has a solution (for a given input $(a_1, \ldots, a_n, b)$ if and only if it has a solution either for $(a_1, \ldots, a_{n-1}, b - a_n)$ or for $(a_1, \ldots, a_{n-1}, b)$. So an instance of the subset sum problem for n numbers can be reduced to two subproblems with $n-1$ numbers each. Building up a search tree based on this observation yields an $O(2^n)$ algorithm (which basically enumerates all subsets of the a_i).

But looking at this tree more carefully, we see that, at least if b is small compared with 2^n, it has a crucial property: *the same subproblem occurs on many branches!* In fact, there are only nb distinct subproblems altogether: for each $c \leq b$ and each $m \leq n$, the subset-sum problem with input $\{a_1, \ldots, a_m, c\}$. Imagine that the branches of the search tree "grow together" if the same subproblem occurs: we get a "search digraph" D. The nodes of this acyclic digraph are labelled with pairs (c, m), and there is an edge from (c, m) to $(c', m-1)$ if either $c' = c$ or $c' = c - a_m$. The

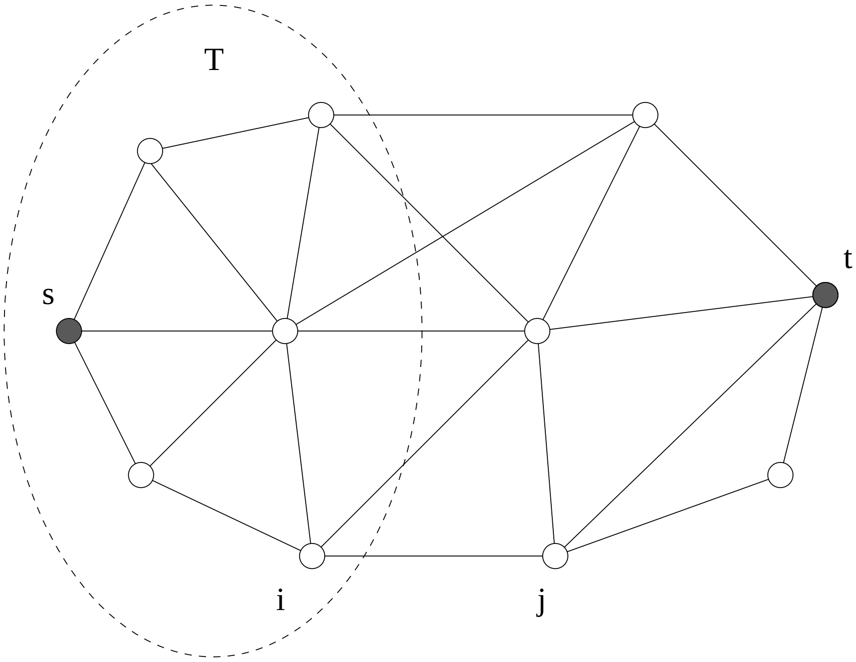

Figure 6: Dijkstra's shortest path algorithm.

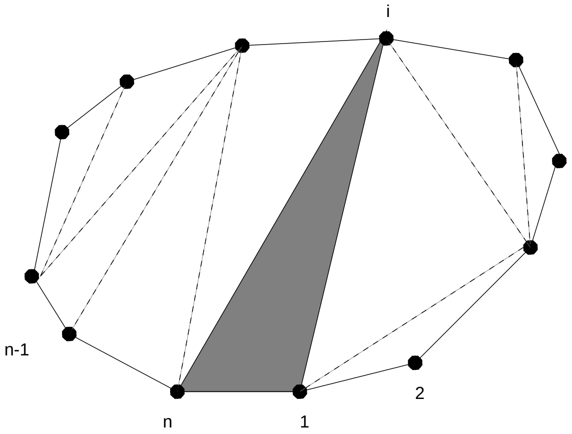

Figure 7: Optimum triangulation of a convex polygon.

subset-sum problem is solvable if and only if there is a dipath from (b, n) to $(0, 0)$. Such a dipath can be found (if it exists) in $O(bn)$ time by searching D either from $(0, 0)$ or from (b, n).

Along the same lines, one can devise an algorithm for the knapsack problem with running time polynomial in $b + \sum_i \langle c_i \rangle$.

While the phrase "dynamic programming" seems to mean different methods in different areas, here we use it to refer to optimization algorithms where a search tree can be found so that the number of subproblems remains polynomial and so a polynomial-time algorithm can be obtained by searching through a "search digraph".

Minimal triangulation of a convex polygon. Given a convex polygon P with n vertices in the plane, we want to find a triangulation with minimal total edge length. (The length c_{ij} of each edge ij is known.)

If the vertices of P are numbered consecutively 1 through n, take edge $1n$ and consider the vertex i with which it forms a triangle in the triangulation. For a given i, it suffices to find optimal triangulations of the polygons with vertices $1 \dots i$ and $i \dots n$, respectively, which can be done independently. So we have produced $2(n-2)$ subproblems.

If we are not careful, repeating this could lead to exponentially many distinct subproblems. But note that if we choose the triangle containing the edge

$1i$ to cut the polygon $1 \dots i$, then we get two subproblems corresponding to convex polygons having only one edge that is not an edge of P.

In general, given two vertices i and j with $i \le j-2$, let $f(i,j)$ denote the minimum total length of diagonals triangulating the polygon with vertices $(i, i+1, \dots j)$. Then clearly $f(i,i+2)=0$ and

$$\begin{aligned} f(i,j) &= \min\Big\{ \min_{i+2 \le k \le j-2} (f(i,k)+f(k,j)+c_{ik}+c_{kj}), \qquad (4) \\ &\qquad f(i+1,j)+c_{i+1,j}, f(i,j+1)+c_{i,j+1} \Big\}. \qquad (5) \end{aligned}$$

The answer to the original question is $f(n,1)$.

We can represent the computation by a "search digraph" whose nodes correspond to all the polygons with vertex set $(i, i+1, \dots, j)$, where $1 \le i, j \le n$, $j \le j-2$. We set $f(i,j)=0$ if $j=i+2$, and can use (4) recursively if $j > i+2$. There are $O(n^2)$ subproblems to solve, and each recursive step takes $O(n)$ time, so we get an $O(n^3)$ algorithm.

Steiner trees in planar graphs. Let $G=(V,E)$ be a graph with edge lengths $c_e > 0$, and let $T \subseteq V$ be a set of "terminals". A *Steiner tree* in G is a subtree that contains all nodes of T. The *Steiner tree problem* is the task of finding a shortest Steiner tree. This problem is NP -complete in general, even for planar graphs. But in the case of a planar graph when all the terminal nodes are on one, say, on the outer face C, a shortest Steiner tree can be found by dynamic programming as follows. Let us first look at a minimum Steiner tree B. Pick any node v of B and let B' be the union of some branches of B rooted at v and that are consecutive in the natural cyclic order of the edges beaving v. Let $T' := T \cap V(B')$. We observe the following:

- There is a path $P \subseteq C$ whose end nodes are terminals such that $T' = V(P) \cap T$.
- B' is a minimum length Steiner tree with respect to the terminal set $T' \cup \{v\}$.
- v is on the outer face of the subgraph $B' \cup P$.

These observations motivate the following dynamic program for the solution of our Steiner tree problem. For every path $P \subseteq C$ whose end nodes are

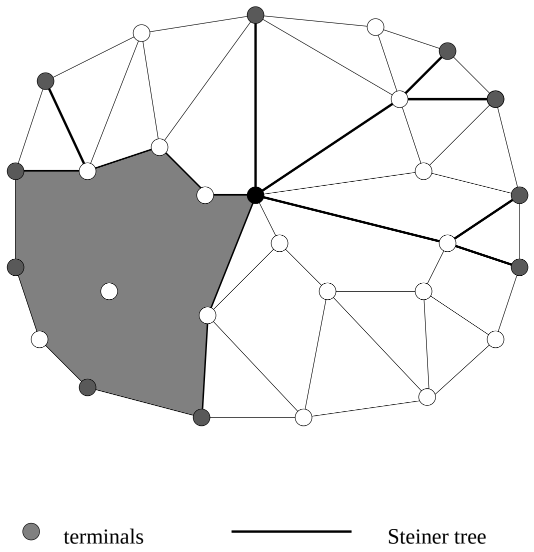

Figure 8: Steiner tree in planar graphs

terminals and every node $v \in V$, we determine a shortest Steiner tree B' with respect to the set of terminal nodes $(V(P) \cap T) \cup \{v\}$ with the additional requirement that v is on the outer face of $B' \cup P$.

If P consists of just one terminal then such a Steiner tree can be found by a shortest path calculation.

Suppose that we have solved this subproblem for all nodes $v \in V$ and all paths $P \subseteq C$ containing at most k terminal nodes. To solve the subproblem for some node $v \in V$ and a path $P \subseteq C$ containing $k+1$ terminal nodes, we do the following. Let $t_1, \ldots, t_{k+1}$ be the terminals contained in P in the natural order. For every node $w \in V$ and every two subpaths P_1, P_2 of P, where P_1 connects t_1 to t_j and P_2 connects t_{j+1} to t_{k+1}, $1 \leq j \leq k$, we solve the subproblems for w and P_1 and for w and P_2 to get two trees B_1 and B_2. We also compute a shortest path Q from w to v. Among all the sets $B_1 \cup B_2 \cup Q$ computed this way choose the one with minimum length. This is an optimum solution of our subproblem for v and P.

To get a minimum length Steiner tree consider a path $P \subseteq C$ that contains all terminal nodes and choose the shortest among all solutions of subproblems for v and P with $v \in V$.

This algorithm is due to Erickson, Monma and Veinott (1987) and is based on ideas of Dreyfus and Wagner. It can be extended to the case when all terminals are on a fixed number of faces.

There are many other non-trivial applications of the idea of dynamic programming; for example, Chvátal and Klincsek (1980) use it to design a polynomial time algorithm that finds a maximum cardinality subset of a set of n points in the plane that forms the vertices of a convex polygon (cf. Chapter 17, section 7.2).

6.1.1 Largest convex subset

Given a finite set S of points in the plane in general position (no 3 points are colinear), we want to find the set $X \subset S$ of largest cardinality and whose points form a convex polygon.

Erdös and Szekeres proved that there exists a function $f(n)$ such that any S of size at least $f(n)$ has a convex subset of size n. It is known that

$$2^{n-2} + 1 \leq f(n) \leq \binom{2n}{n},$$

and based on the values of f for 3,4 and 5 it is conjectured that the lower bound is tight (though there is not yet a proof even for $f(6) = 17$).

However, Chvátal and Klincsek showed that given an instance of this problem, we can find X in polynomial time using dynamic programming. If we know already part of the convex polygon formed by X, say $x_1, x_2, \ldots x_i$, in order to finish building X in an optimal way we only need to carry over the value of the first and last two points (so that the slopes of the end edges are known). Thus if $F(x, y, z, u)$ is defined to be the maximum number of vertices in a convex polygon containing x, y, z, u consecutively, there is a recurrence giving F:

$$F(x, y, z, u) = \begin{cases} \max_{\{v : xyzuv \text{convex}\}} F(x, y, u, v) + 1 & \text{if such a } v \text{ exists} \\ 4 & \text{otherwise.} \end{cases}$$

Since z is then eliminated from further consideration, there cannot be more than n steps in the recursion, and thus the algorithm finishes. The computations can be represented on a table where the i^{th} column contains the $xyzu$ for which $\{v : \quad xyzuv \text{ convex}\}$ has cardinality $i-1$. The table has size $O(n^4)$ and each computation takes time $O(n)$, hence an $O(n^5)$ algorithm.

Actually, there are much better ways to solve the problem; the best known algorithm runs in $O(n^2)$.

Given two points x and y, another solution computes the longest sequence of vertices that form a concave path going from x to y, staying above xy, and stricly increasing along the x-coordinate. Given x,y and z, the last vertex on the path before y, the longest sequence can be computed recursively as a function of all the longest paths from x to z with last vertex u (and which are concave, above xz and increasing in the x-direction). This gives a new, better dynamic programming algorithm which can be used for computing the largest convex subset.

In higher dimensions, the complexity of this problem is unknown.

In dimension 2, we see that the algorithm is easily adapted to compute the convex subset with largest total weight, or the largest convex subset which contains no interior points.

Finally, we present an application of dynamical programming to solving an enumeration problem.

6.1.2 An enumeration problem

Given the alphabet $\Sigma = \{a, b, c\}$, we want to compute the number $F(n)$ of words on the alphabet, of length n, and which contain no subword of the form *aa*, *bbb* or *bbc*.

Up to some correcting terms, we expect $F(n+1)$ to be close to $3F(n)$. Extending a word already constructed can be done in different ways depending on how it ends. Thus we define $F_{uv}(n)$ to be the number of words of length n ending in uv and containing no *aa*, *bbb* or *bbc* subword. There are 9 functions to determine.

The recurrence relations are easy to establish:

$$\begin{array}{rcll} F_{aa}(n+1) & = & 0 & \\ F_{bb}(n+1) & = & F_{ab}(n)+F_{cb}(n) & \\ F_{bc}(n+1) & = & F_{ab}(n)+F_{cb}(n) & \\ F_{uv}(n+1) & = & F_{au}(n)+F_{bu}(n)+F_{cu}(n) & \text{otherwise.} \end{array}$$

A standard way of solving this system of equations is by the use of a matrix. The 9-coordinates vector $F^{(n)} = (F_{uv}(n))$ is defined by recurrence by $F^{(n+1)} = MF^{(n)}$, where M is a square 0-1 matrix defined from the 9 equations. Thus $F^{(n)} = M^{n-2}F^{(2)}$, and $F(n)$ is the sum of the coordinates of $F^{(n)}$.

In the favorable case, where M can be diagonalized, $M = T^{-1}\Delta T$, where Δ is the diagonal matrix of the eigenvalues, we get a solution of the form $F^{(n)} = \tilde{a}\Delta^{n-2}\tilde{b}$, where $\tilde{a}$ and $\tilde{b}$ are constant vectors, and so

$$F(n) = \sum_{1\leq i\leq 9} (a_i b_i){\lambda_i}^{n-2} \ .$$

This explicit formula tells us a lot about the function. For example, we can see that the largest eigenvalue λ_1 eventually dominates, and so $F(n)$ is asymptotic to $(a_1b_1)\lambda_1^{n-2}$.

In the unfavorable case, where M is not diagonalizable, we can put it in Jordan normal form and still get a closed formula for $F(n)$, where polynomials in n replace the constant coefficients of the ${\lambda_i}^{n-2}$ terms. Details can be found in any basic linear alebra book.

As an exercise to apply this basic method, the reader may want to think about the following problem: find the number of permutations $(a_1a_2\ldots a_n)$ of $\{1\ldots n\}$ such that $\forall i \quad |a_i - i| \leq 2$.

In the next section, we are going to see a general framework in which dynamic prograamminf type algorithms turn out very powerful.

6.2 Tree-decomposition of a graph

6.2.1 Dynamic programming on a tree

Many NP-complete problems are easy when restricted to trees; a typical algorithm to exploit the special structure of trees is the following.

Definition 6.1 *Let G be a graph and $c: V \rightarrow \mathbb{R}_+$, a weighting of its nodes. The* stable set problem *is to find a stable subset of V (i.e., a subset no two elements of which are adjacent) with maximum weight.*

The stable set problem is NP-hard. For trees, however, the following sinple algorithm works.

Let r be any root. For every node v, let T_v be the granch of G rooted ad v, i.e., the set of all nodes u such that the unique $u - r$ path is G contains v. Let $f(v)$ denote the maximum weight of a stable set in T_v containing the node v, and $g(v)$, the maximum weight of a stable set in T_v not containing node v. Let $v_1, \ldots, v_k$ be the neighbors of v. Then

$$f(v) = c(v) + \sum_{i=1}^{k} g(v_i),$$

$$g(v) = \sum_{i=1}^{k} \max\{f(v_i), g(v_i)\}.$$

These recurrences allow us to determine the value of $f(v)$ and $g(v)$ for every v (working from the leaves to the root). The solution of the original problem is $\max\{f(r), g(r)\}$.

Exercise 6.1 *Work out a similar algorithm to find the maximum weight of a set of nodes in a tree such that the distance of any two is at least 3.*

Can this method be extended to more general graphs? One way is to define "tree-like" graphs. A general framework to do so was developed by Robertson and Seymour in their "Graph Minors" theory. In the following we give a glimpse of this theory, and how it can be applied to algorithm design.

6.2.2 Tree-docomposition and tree-width: definitions

We define the important graph theory notion of tree-decomposition and show how it can be used to find dynamic programming type algorithms for solving in polynomial time restricted versions of NP-complete problems.

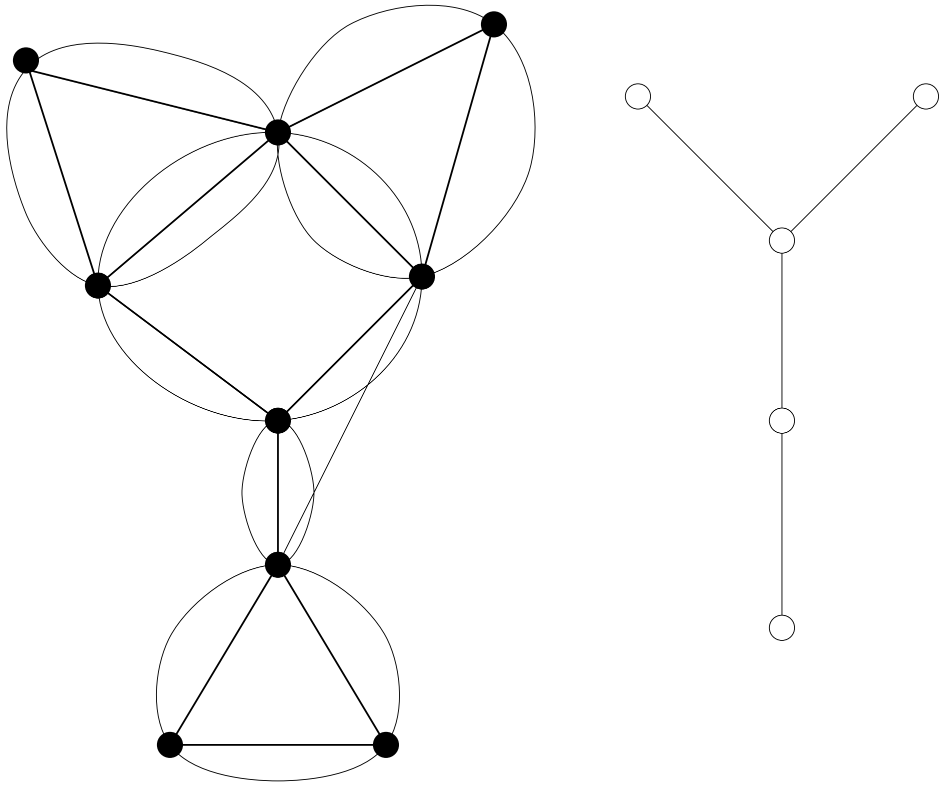

Figure 9: A tree-decomposition of a graph.

A *tree-decomposition* of a graph G is given by a tree T and a family $(H_i)_{i \in V(T)}$ of subgraphs of G such that $G = \cup_i H_i$ and the following property holds: if vertex i is on the path from j to k in T, then $V(H_i) \supset V(H_j) \cap V(H_k)$.

An equivalent property is that for every vertex v of $V(G)$, the set $\{i \mid v \in V(H_i)\}$ spans a subtree of T. (It is easy to see that the two properties are equivalent). An example of a tree-decomposition can be seen on figure 9. Of course, there are many possible tree-decompositions of a graph; for example, we can take T to have one node i and $H_i = G$. Robertson and Seymour have introduced a useful concept related to tree-decompositions. The *tree-width* of a graph G is the minimum integer p such that there exists a tree-decomposition of G, $G = \cup_i H_i$, for which no part has size greater than $p+1$: in other words, $\forall i \;\; |V(H_i)| \leq p+1$.

Let us illustrate the notion of the tree-width by some examples. (Computing the tree-width of a graph is NP-complete in general.)

Lemma 6.1 *G has tree-width ≤ 1 if and only if it is a forest.*

For a less obvious example, we need a definition. A *series-parallel* graph is defined by induction: any graph with no edges is series-parallel; any series-parallel graph can be extended in one of the following four ways (see figure 10):

1. add a new node and connect it to an old node by a new edge;

2. add a loop;

3. add an edge parallel to an edge already present.

4. subdivide an edge by creating a node in the middle.

Lemma 6.2 *G has tree-width ≤ 2 if and only if it is series-parallel.*

Proof *(sketch)*: The lemma can be proved by induction. Assume that G is a series-parallel, then it is obtained from a series-parallel graph G' by one of the operations above, and we may assume that the graph G' has tree-width at most 2. For operations (2) and (3), we can extend the decomposition by adding a new edge without changing the decomposition; in the other two cases we can add a part to the tree decomposition, containing the new vertex and its one or two neighbours, and the tree-width is still 2. Since the initial series-parallel graphs have tree-width ≤ 2, it follows that all series-parallel graphs have tree-width at most 2.

Conversely, if we take a graph with tree-width 2, considering its tree-decomposition, we take a vertex in an end-part (i.e. a part which corresponds to a leaf of tree T); if it has only one neighbour then it could have been

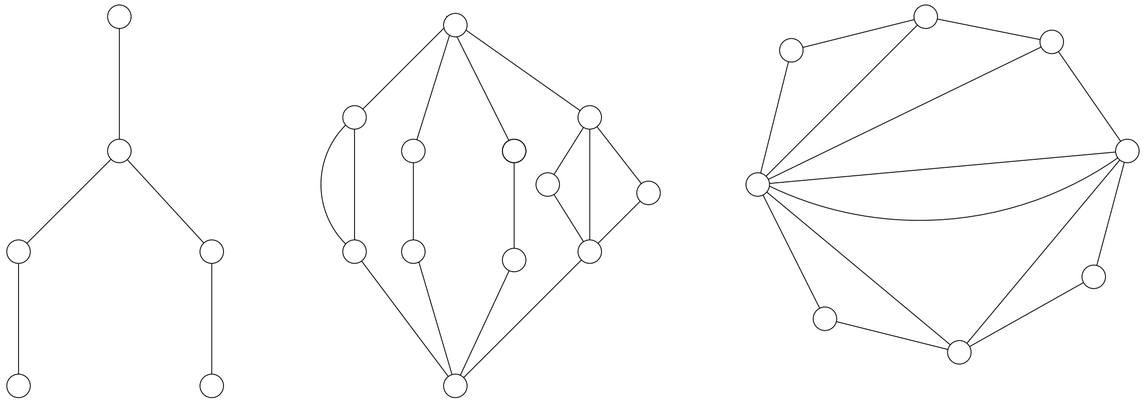

Figure 10: Series-parallel graphs.

obtained by (1), (2) or (3); and if it has two neighbours then it may come from subdividing an edge between the two neighbours. Thus we can delete the vertex, and the new, smaller graph still has tree-width at most two and is series-parallel by induction. ∎

Another important notion that plays role here is the following. the *contraction* of edge uv deletes vertices u and v and replaces them with a new vertex $\overline{uv}$ whose neighbor set is the union of the neighbor sets of u and v). G' is a *minor* of G if G' arises from G by a sequence of edge-contractions and edge and vertex-deletions. (See figure 11).

It can be seen that the notions of minors for graphs and for matroids match. Many graph-theoretic results use this notion; a very typical one is Kuratowski's Theorem:

Theorem 6.1 *A graph is planar if and only if it does not contain* K_5 *and* $K_{3,3}$ *as a minor. (See figure 12.)*

Using this notion, we can give further characterizations of graphs with small tree-width:

Lemma 6.3 *A graph has tree-width at most* 1 *iff it has no* K_3 *minor. A graph has tree-width at most* 2 *iff it has no* K_4 *minor.*

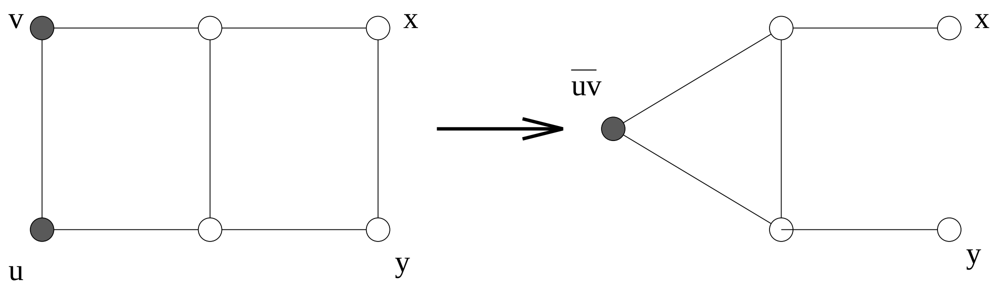

Figure 11: Contraction of edge uv and deletion of edge xy.

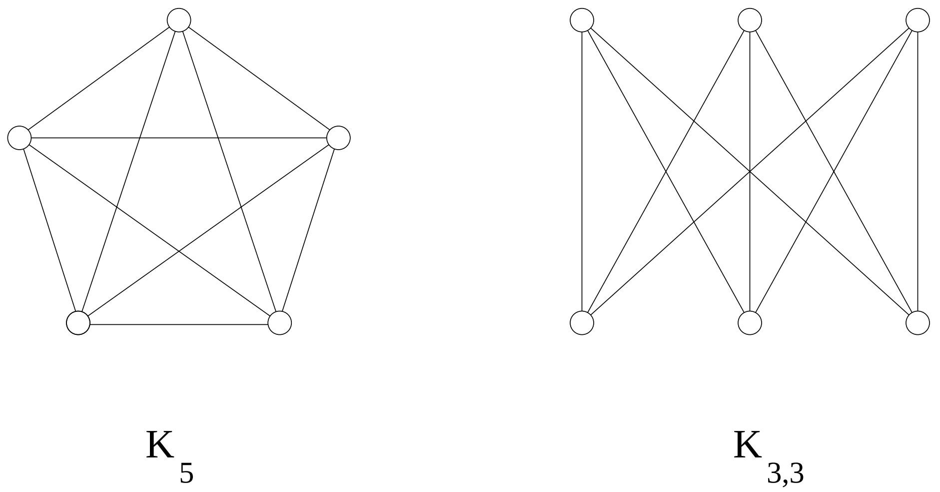

Figure 12: Kuratowski's graphs

Unfortunately, this does not carry over for tree-width 3.

Exercise 6.2 *Show that an $n \times n$ "grid" has arbitrarily large tree-width.*

But the following difficult theorem of Robertson and Seymour does give a deep connection between treewidth and minors.

Theorem 6.2 *Let H be a planar graph. If G has no H-minor, then the tree-width of G is bounded by $f(H)$.*

We will not prove this difficult theorem. It is worth pausing for a moment here to think about the meaning of this result. Given H, $f(H)$ is a fixed constant, but as H varies the only upper bound known for $f(H)$ is exponential in $|V(H)|$ (though it is not known whether the bound is tight).

The result can also be viewed as a week version of a (nice but false) min-max theorem: *the maximum size of a grid minor of a graph is equal to its tree width.* What this theorem and the exercise does imply is that *the tree-width of a sequence of graphs tends to infinity if and only if the size of their largest grid minor tends to infinity.*

We see that the theorem cannot be true for non-planar graphs H: for example if H is K_5 (the complete graph on 5 vertices), then no grid, however large, can have H as a minor. Yet the theorem relating H-minors to tree-width has a generalization to arbitrary excluded minors; but this is much more complicated, and even to state it precisely would take several hours. Assume that G has no H-minor, where H is an arbitrary fixed graph. How can that happen? There is a theorem of Robertson and Seymour that describes the structure of those graphs containing no H minor.

One possible cause may be that G is just too small to contain H. Another, more sophisticated reason may be that G is simple enough to be embeddable on a certain surface F on which H cannot be embedded. In this latter structure, there is a rather technical modification that can be carried out: we may add to G some "fringes", i.e. on a large face in this embedding we can add diagonals all along the boundary, with the endpoints of each diagonal at a bounded distance from each other along the boundary (this annoying particularity seems to spoil the whole structure...)

Now Roberston and Seymour prove that every graph G not containing an H minor has a tree decomposition into parts which can be embedded in some fixed surface (depending only on H), with fringes, with a bounded number of points added and connected arbitrarily to the rest.

We do not state the general theorem formally, but one should be aware of its existence. It has some important applications in complexity theory, as we shall see in the next sections.

6.2.3 Chromatic number of graphs with small tree-width

Definition 6.2 *The* chromatic number $\chi(G)$ *of a graph* G *is the minimal number of colors with which* $V(G)$ *can be colored in such a way that no two adjacent vertices have the same color.*

The famous Four Color Theorem states that $\chi(G) \leq 4$ for planar graphs. In general, deciding whether $\chi(G) \leq 3$ is NP-complete (though deciding whether $\chi(G) \leq 2$ is very easy). We are going to see that if G has tree-width at most p, where p is a constant, then there is a polynomial time algorithm for deciding if $\chi(G) \leq k$ (and hence for computing the chromatic number).

We assume that the tree-decomposition of G is given (that by itself is no trivial matter, so it is a seriously limiting assumption...). First we can see that if $k > p$ it is easy to find an admissible coloring: take a vertex x of an end-part of the decomposition; delete it temporarily, color the rest of the graph by induction; x has at most p neighbors and can therefore be colored.

If $k \leq p$, we use the dynamic programming approach. Take a part H_i of the decomposition and call it "root". Color H_i with k colors (there are at most k^{p+1} possibilities) so that no two neighbors in H_i have the same color. H_i defines a number of "branches": deleting i in T partitions the tree into several connected components, and a branch of G is by definition the union of all the H_j's such that j spans one of the connected components of T (see figure 13).

Each branch is smaller than the original graph, and its root is partially colored. Assuming that we know already which admissible colorings of

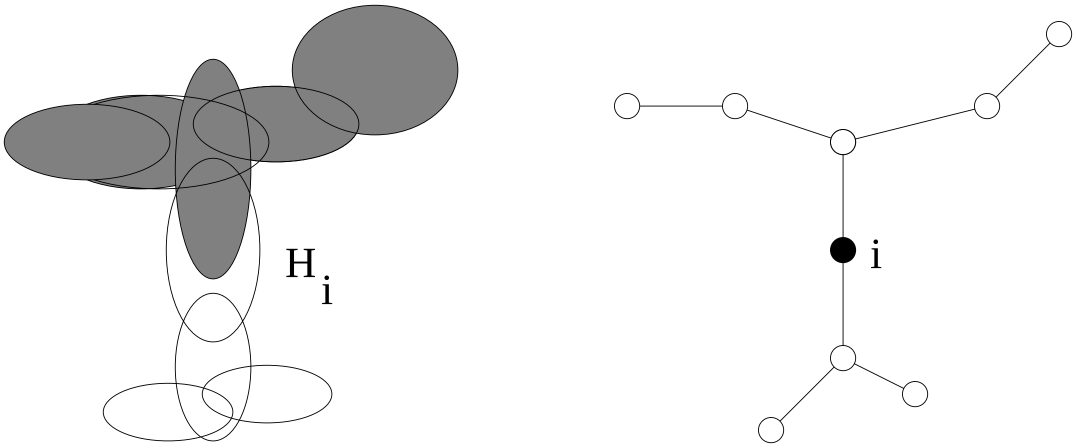

Figure 13: A branch of a tree-decomposition of G.

their roots can be extended to admissible colorings of the branches, check for each branch if it has an admissible coloring which matches the coloring of H_i. If the coloring of H_i can thus be extended to all the branches, then it extends to their union (by the definition of tree-decomposition: two branches don't intersect outside H_i), yielding a coloring of the whole graph.

Thus, we can construct a dynamic program, representing the computations in a table. The first column corresponds to the branches with only one end-piece (these branches correspond to the leaves of T), together with a coloring of that piece; the second column deals with the branches which correspond to one edge of T (the endpart and its neighbour), with a coloring of the root (i.e. of the part which is not the endpart), and so on. Each edge of T defines exactly two branches, so there are at most $2n$ branches to consider, together with colorings of their root. Since there are at most k^{p+1} admissible colorings for a root, the table has at most $2k^{p+1}n$ entries.

Each entry is 0 or 1 telling whether the given coloring of the root extends to a coloring of the whole branch. So we have all 1 in the first column. The computation of a new entry of the table, corresponding to branch B

and coloring α of its root, requires looking at every sub-branch B' from the root of B (at most n) and at every coloring α' of the root of this sub-branch which is compatible with α. We write a "1" iff for every B' there is an α' such that the corresponding entry in the table (which has been filled out earlier) is 1.

Thus the running time of the algorithm is at worst $O(n^2)$ with a constant factor depending on p.

This shows that even once the concept of dynamic programming is understood, applying it correctly is not always trivial...

6.2.4 Other applications of tree-decompositions

We sketch other applications of tree-decompositions. If we understand the idea of the previous proof, we see that the weighted stable set problem (and many other NP-complete problems) can be solved for graphs with bounded tree-width in polynomial time by the same scheme.

One important example of such a problem is the following.

Definition 6.3 *Given a graph G and vertices $u_1, \ldots, u_k, v_1, \ldots, v_k$, the* linkage problem *asks if there exists vertex-disjoint paths in G linking u_1 to v_1, u_2 to v_2, ... and u_k to v_k. In general, this is an NP-complete problem, but for graphs with bounded tree-width it can be solved in polynomial time ($O_k(n^3)$ or so).*

(People familiar with graph theory should not be confused by the similarity with Menger's theorem: one version of Menger's problem asks for paths linking $\{u_1, \ldots u_k\}$ to $\{v_1, \ldots, v_k\}$ but in an *unspecified* order, and flow methods then give a polynomial-time algorithm, as we shall see later.)

The problem has applications for VLSI routing, but is NP-complete even if the underlying graph is planar.

If such paths do not exist in G, one reason may be because of "cuts": they are sets of $i-1$ vertices which separate a subset of i vertices in $\{u_1, \ldots u_k\}$ from the corresponding subset in $\{v_1, \ldots v_k\}$. But even if there are no such cuts this still does not guarantee the existence of links: see figure 14 for an example of a planar graph, 5-connected, but which cannot connect $\{u_1, u_2\}$

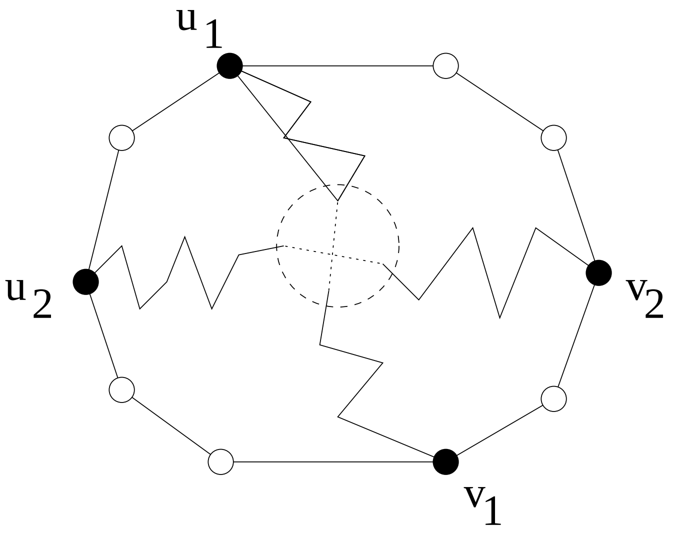

Figure 14: A planar graph with no good linkage.

to $\{v_1, v_2\}$ with disjoint paths. The argument is inherently topological (hence sophisticated in this framework).

But if we assume that the tree-width is bounded, then a dynamic programming approach gives a polynomial time algorithm. (Tree-widths are really a powerful notion!) Using much more from the Robertson-Seymour theory, it also follows that for fixed k, the linkage problem can be solved in polynomial time. It also implies that for every fixed H, there is a polynomial time algorithm to check whether an input graph G has an H minor. The presentation of these results, however, would take at least a whole course.

Perhaps the most dramatic consequence of the Robertson-Seymour theory is that if P is any property of graphs that is inherited to minors (e.g., embeddablity in a surface, not containing a given minor etc.) then P can be decided in polynomial time. (I don't know of any other "positive" result on algorithms of this depth and generality.)

7 Augmenting paths

We have discussed the strategy of finding a solution to a combinatorial optimization problem and then looking for local improvements. Sometimes one can use quite involved methods to find such improvements, because the better solution we try to find is not "nearby" in any easy sense. If this is the case it is more common to refer to an improving step as an "augmentation" rather than a "local improvement". The most important augmentation method is the method of augmenting paths, where the graph (or other combinatorial object) is searched through to find an augmentation. The most important applications of this method are to mathings, flows and matroids.

7.1 Maximum matching and alternating paths

Let G be a graph; we want to find a maximum size matching in G (recall that a *matching* is subset of the edges having mutually no endpoint in common). A reference for matching theory is L. Lovasz – M. D. Plummer, Matching Theory, North-Holland 1986.

Assume that we have a matching M; how can we see whether M is maximum? One answer to this question is provided by the following simple criterion due to Berge. An *alternating path* (with respect to M) is a path in which every other edge belongs to M (the path may start with an edge of M or with an edge not in M). An alterning path is *augmenting* if it starts and ends with an edge not in M, and moreover, its endpoints are not incindent with any edge of M. (An augmenting path has necessarily odd length.)

Fact 7.1 *A matching in G is maximum if and only if there is no augmenting path with respect to it.*

Proof. It is obvious that if we have an augmenting path P with respect to M, then we can improve M: we can replace in M the edges of $P \cap M$ by the edges of $P \setminus M$, to get a larger matching.

Conversely, if M is not maximum, and there exists a larger matching M', then consider the connected components of $M \cup M''$. These are either alternating paths or alternating circuits with respect to M. At least one such component must contain more edges of M' than of M: this coponent is an augmenting path with respect to M. ∎

This fact is not very deep since it relates the existence of something (a larger matching) to the existence of something else (an augmenting path). But it does give a motivation for most of the known polynomial time matching algorithms: one can look for an augmenting path by building up a search tree. This idea works out very simply for bipartite graphs, and we start with describing this case.

7.2 Maximum bipartite matchings

We describe an algorithm to fing a maximum matching in a bipartite graph. The presentation may be more complicated than necessary, but it will generalize nicely to general graphs.

Given a bipartite graph $G = (A, B)$, we want to find a maximal matching M. We use the notion of *alternating trees*. Given a matching M, we call a vertex of the graph *exposed* (by M) if it is not covered by M. An *alternating tree* is a subgraph of G which contains exactly one exposed node r; for which every node at odd distance from r has degree 2 in the tree; such that along any path from r every other edge belongs to the matching; and whose endpoints are all at even distance from r.

The vertices of the tree at even distance from r (including r itself) are called *outer* nodes, and the vertices at odd distance from r are called *inner* nodes.

An *alternating forest* is a forest such that every component is an alternating tree, and every exposed node of the graph is contained in the forest.

The notion of alternating trees is very natural. If we have a matching M and would like to extend it, we take an exposed vertex r and try to match it to a neighbour; but the neighbor may already be in the matching, so we leave this edge out; this creates a new exposed node which we try to match etc. This leads to the notion of alternating paths. Searching for alternating paths from an exposed node r leads to the construction of an alternating tree.

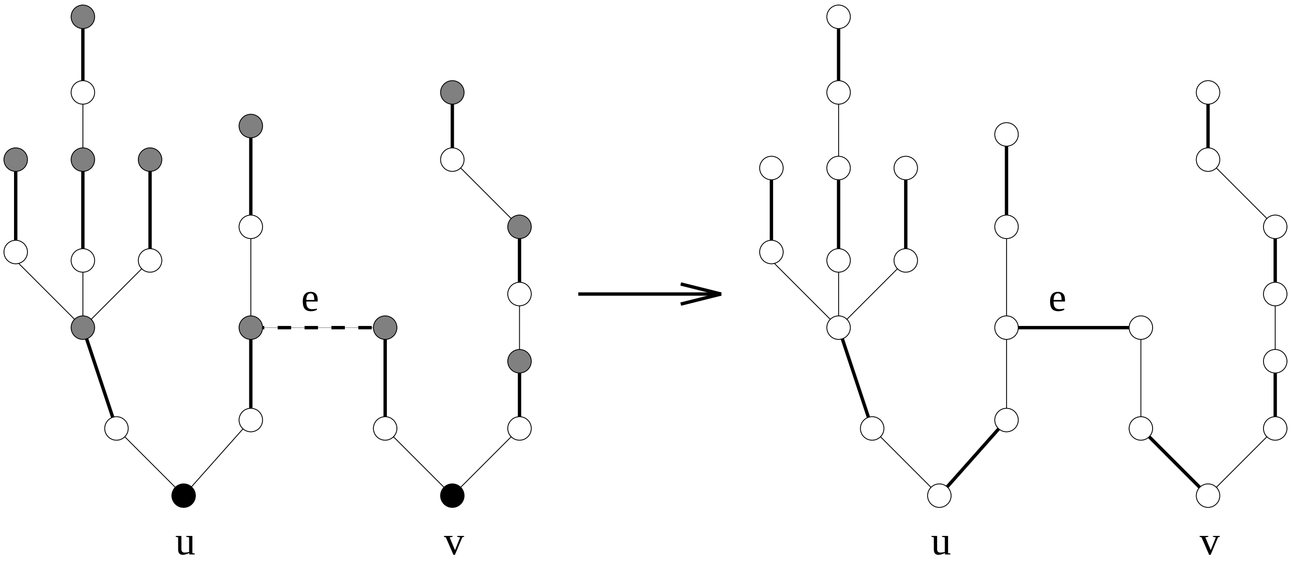

Figure 15: Extending a bipartite graph matching

The algorithm proceeds step by step. At the beginning, we have the empty matching and the alternating forest consisting of all nodes and no edges. At every stage, either we can extend the matching to a new matching with more edges, or we can extend the forest already constructed for the matching under study, or we are stuck and the matching is optimal.

Suppose that we already have a candidate matching M and alternating forest F with respect to M.

Case 1. If there is an edge e connecting two outer nodes of the forest, then these nodes must belong to different trees since the graph is bipartite. Let u and v be the exposed nodes of the two trees. We have an alternating path from u to v, and switching the matching edges along the path we get a new matching M' with one more edge than M. The forest consisting of all exposed points of M' is alternating with respect to M'. (See figure 15).

Case 2. Otherwise, if there is an edge connecting an outer node p of the forest to a vertex u of $V(G) \setminus V(F)$, this vertex u cannot be exposed from the definition of an alternating forest. Hence it is the endpoint of an edge

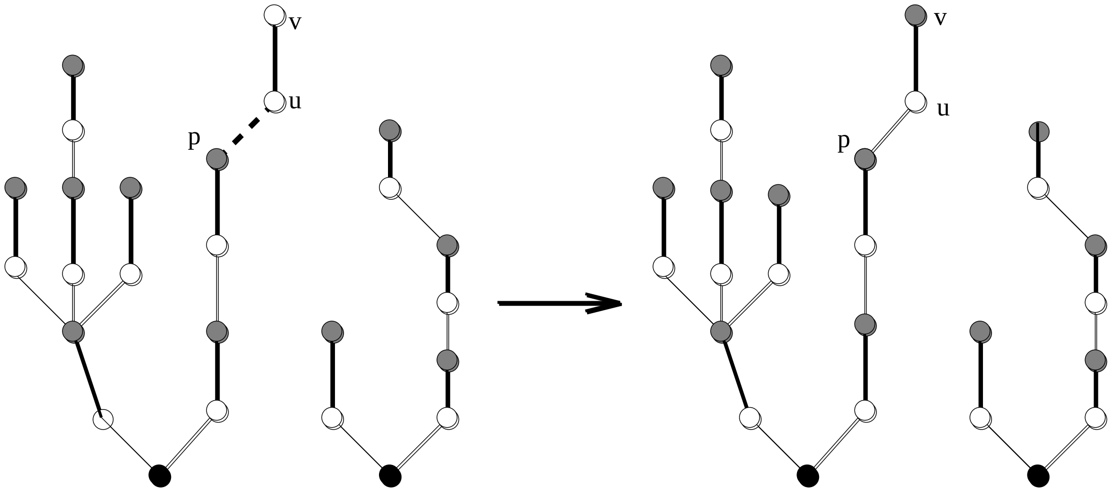

Figure 16: Extending an alternating forest

of the matching, say uv. Vertex v is also in $V(G) \setminus V(F)$ (otherwise u would have to be in F), and so we can extend F by adding the edges pu and uv. (See figure 16).

Case 3. (case 3) If neither of the cases above applies, the outer points must be connected to inner points only. A simple but crucial counting argument will show that the matching must be optimal. (We will use this counting argument several times during this lecture).

There is one exposed vertex in every component of F, and none outside F, so the total number of exposed points of the matching is equal to the number of components of F. Each outer point of the trees of F is matched to an inner point, which is its predecessor in the tree, except for the exposed vertex of each tree. Thus

$$\#\text{outer points } = \#\text{inner points } + \#\text{exposed points.}$$

Let D be the set of outer vertices and A be the set of inner vertices of F. The neighbor set of D in the graph is included in A, the number of exposed points of M is exactly $|D| - |A|$.

Let M' be any matching of the graph. M' can only match points of D to points of A, and must therefore it must leave at least $|D| - |A|$ points exposed. Hence M is a maximum matching.

Thus we have a valid algorithm for finding a maximum matching in a bipartite graph. At every step we must consider at most n^2 edges, and since the size of M or of F increases at each step, there are at most n^2 steps, hence a $O(n^4)$ algorithm. Simple considerations enable to reduce the running time to $O(n^3)$. Tricky implementation details reduce it further to $O(n^{2.5})$.

7.3 General maximum matchings

We now turn to the more difficult problem of finding a maximal matching in a general graph G. A similar algorithm is used. We assume that we have computed a matching M and an alternating forest F, and we try to enlarge either F or M just as in the bipartite case. The only point where we used the fact that the graph was bipartite was to argue that an edge connecting two outer points must connect points of different trees. Thus we have to deal with an extra case:

Case 1'. There exists an edge connecting two outer points in the same component of the forest. This edge must close an odd-length cycle in the alternating tree. We contract this odd cycle C to a single point. In the new graph G', the matching M has become M', which is still a matching (there was at most one matching edge between C and the rest of the graph). The forest F becomes a new forest F', which is still alternating with respect to M' (see figure 17). Also note that M' leaves exactly as many nodes exposed in G' as M does in G. Let $2k + 1$ be the number of vertices of C. Then $|M'| = |M| - k$.

Lemma 7.1 *M is maximum in G if and only if M' is maximum in G'.*

We will not show the $\Longleftarrow$ direction, which is the difficult part of the lemma (in fact it is not true for all cycles, but only for cycles obtained from the alternating forest construction). With a trick, we shall not need this part (in fact, it will follow from our analysis).

We show the $\Longrightarrow$ implication. Assume that there exists a matching ${M_1}'$ in G' such that $|{M_1}'| > |M'|$. We want to exhibit a matching M_1 of G for which $|M_1| > |M|$. We notice that there is at most one edge of ${M_1}'$ going into C. When we lift up the graph to G, we go around the cycle C, picking every

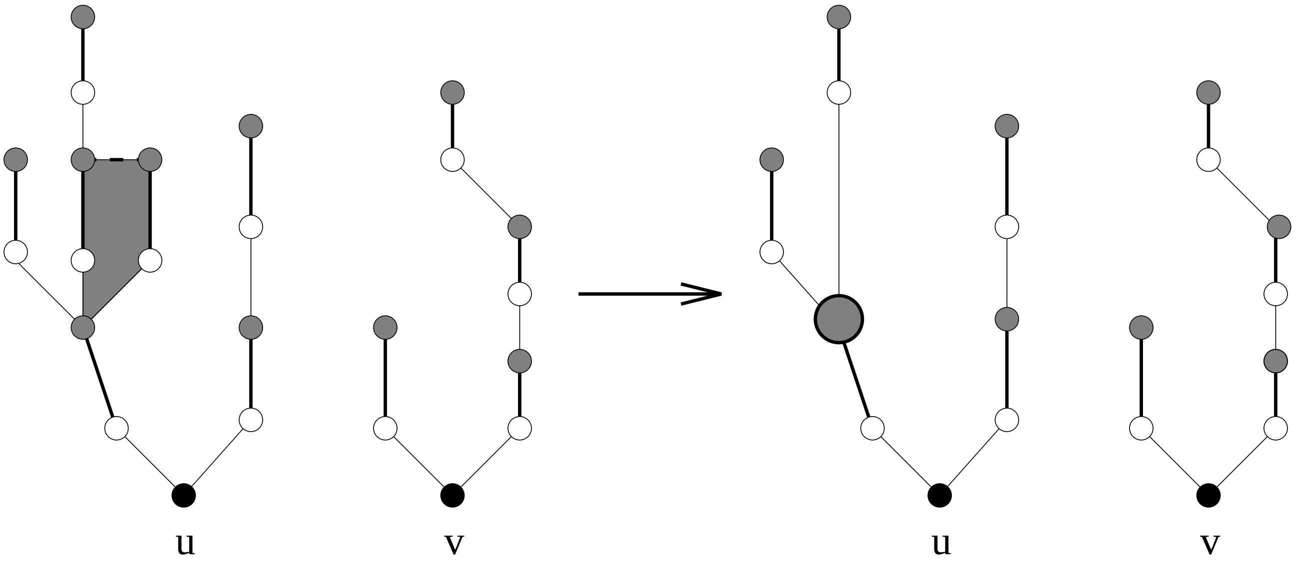

Figure 17: Contracting a cycle

other edge. Adding these edges to the edges of ${M_1}'$, we get a matching M_1 such that

$$|M_1| = |{M_1}'| + k > |M'| + k = |M| \ ,$$

hence the lemma.

We now have all the elements to construct the algorithm. At every step we deal with graph H, which has been obtained from G by a sequence of contractions, together with a matching M of H and an alternating forest F with respect to M.

If case 1 occurs then we lift up the new matching from H to G, using the construction in Lemma 8.1, thus increasing the size of the original matching of G; with this larger matching, we start to grow a new forest from the trivial forest consisting of the exposed nodes and no edges. Case 2 is dealt with as in the bipartite case. In case 1' we contract the resulting odd cycle in graph H with the corresponding transformations on M and F. Eventually the algorithm stops on an occurrance of case 3. We want to prove that if we lift the resulting matching up to the original graph G, we get a maximum matching of G.

Let A [D] be the set of nodes of G that are mapped on inner [outer] points of the final alternating forest F, and let $C = V(G) - A - D$. H was obtained from G by a sequence of contractions of odd-length cycles. We note that whenever we do a contraction, the resulting point is an outer point. Thus the inner points of F are vertices of the original graph, while the outer points of F may correspond to large bits of G, but these bits all have an odd number of points (since each contraction contracts an odd number of vertices into 1 vertex). Since the outer nodes in F are not connected by edges in H, these “bits” are just the connected components of the subgraph of G induced by D, and all these components are odd.

Since the number of exposed points remains constant during the whole procedure of contracting cycles, the counting argument lifted up to G states that

$$\#(\text{components of } D) - |A| = \#(\text{exposed points}).$$

If we consider an arbitrary matching M_1 of G, for each (odd) component of the subgraph of G induced by D, either there is an edge connecting it to an inner point, or there is an exposed point of M_1 inside the component

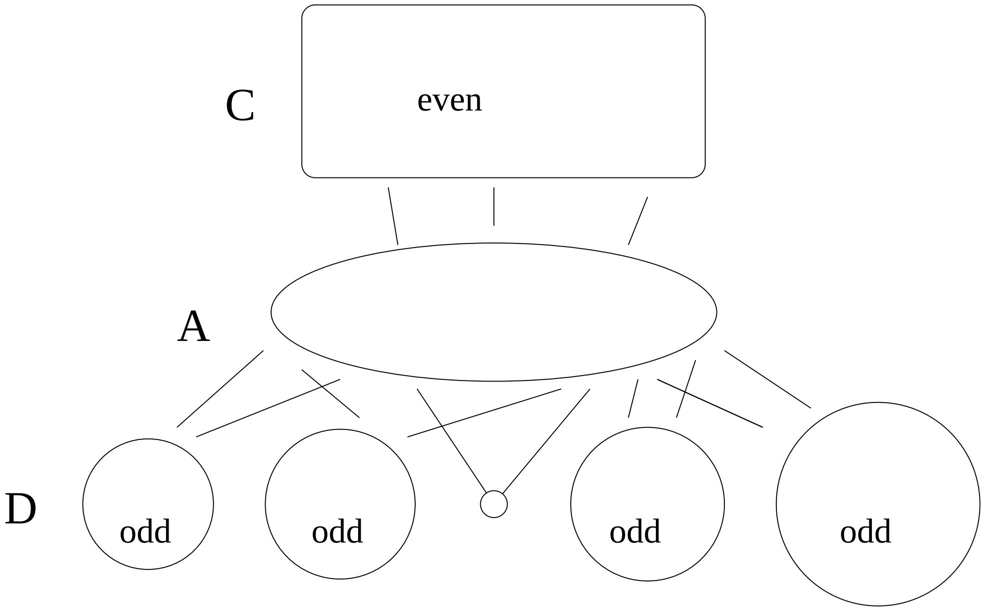

Figure 18: Partition of the vertices of G

(or both). Thus M_1 must have at least as many exposed points as M, and so M is a maximum matching.

We have constructed a valid algorithm for building a maximum matching. A rough analysis of the running time shows an $O(n^4)$ bound. Even and Kariv, and Micali and Vazirani have managed to bring it down to $O(n^{2.5})$.

7.4 Gallai-Edmonds decomposition of a graph

This algorithm gives a lot of information about the graph besides constructing a maximum matching. In particular, it partitions the vertices of G into three classes, D, the set of vertices that became "outer" vertices at the end of the algorithm, A, the set of "inner" vertices, and C, the set of remaining vertices (see figure 18).

We make the following important observation:

Lemma 7.2 *A, C and D are independent of the choices made during the algorithm.*

In fact, there is an equivalent "invariant" definition of D: D is the set of vertices v that are exposed by some maximal matchings of G.

It is easy to see that fact: no point of A can be exposed by a maximum matching, because for a maximum matching, we have equuality in the counting argument. Let t be the number of components spanned by D. If M is a maximum matching, then the number of exposed points is $t - |A|$. But M already has $t - |A|$ exposed points in D. Hence no point outside D can be exposed.

On the other hand, every point of D is exposed by some maximal matching: take an outer point of the first F. By switching the matching edges in the tree from the root to that point, you get a matching which exposes the point, and when lifting things up from H to G we can maintain the fact that a given point is exposed; hence eventually we can choose any point of D.

Thus D is independent of the choices made during the algorithm. A is the neighbor set of D, and so is also uniquely defined, as well as C, their complement in $V(G)$, and the lemma is proved.

Historically, this decomposition was first discovered by Gallai, and independently as a consequence of the algorithm by Edmonds.

Let us point out a few other interesting facts about this decomposition. Given the decomposition and a maximal matching M, every component of D has a near-perfect matching (only one point not matched in D), and from the counting argument there are exactly $|A|$ edges of M from D to A; these edges cover A, and so C, which has no exposed point and no edges going into A, has a perfect matching. In particular, every component of C is even and every component of D is odd.

As a corollary, we get Tutte's theorem:

Theorem 7.1 *G has a perfect matching if and only if*

$$\forall X \cup V(G),\ \#(\textit{odd components in } G \setminus X) \leq |X| \ .$$

Note that these are respectively the NP and co-NP descriptions of the problem.

First we prove the $\Longrightarrow$ direction, which is easier. Given X, $G \setminus X$ is partitioned into several odd and even components. From each odd component, there is at least one edge going to X (otherwise there would not be a perfect matching of the graph), hence the inequality.

Next we prove the $\neg \Longrightarrow \neg$ part of the theorem. If the graph does not have a prefect matching, the algorithm ends up with sets D, C, A. Taking $X = A$ yields a decomposition which contradicts the inequality. Hence the proof.

The theorem can be generalized, giving what is known as the Berge formula: **REFERENCE**

Theorem 7.2

$$\min\{\#\textit{exposed points in } G\} = \max_X \{(\#\textit{odd components of } G \setminus X) - |X|\}.$$

7.5 The Matroid Intersection Problem

We discuss a problem in matroid theory that is a common generalization of the branching and bipartite matching problems and is one of the most general situations in which the method of alternating paths can be successfully applied.

Consider the following problem: Given two matroids $(E, \mathcal{M}_1)$ and $(E, \mathcal{M}_2)$ with the same underlying set, find $\max\{|A| : A \in \mathcal{M}_1 \cap \mathcal{M}_2\}$.

Our first thought is that perhaps $\mathcal{M}_1 \cap \mathcal{M}_2$ is a matroid. A simple counterexample is

$$\mathcal{M}_1 = \{12, 13, \ldots\} \text{ and } \mathcal{M}_2 = \{12, 23, \ldots\}.$$

The intersection $\mathcal{M}_1 \cap \mathcal{M}_2 = \{12, 1, 2, 3, \emptyset\}$ does not satisfy the exchange property. (Consider 12 and 3.) In particular, the greedy algorithm will not solve the problem of finding the largest common independent set in two matroids. In fact, many hard problems can be phrased as a weighted matroid intersection problem.

We can also formulate a weighted version of this problem: Given two matroids $(E, \mathcal{M}_1)$ and $(E, \mathcal{M}_2)$ with the same underlying set and $w : E \to \mathbf{R}_+$, find $\max\{w(A) : A \in \mathcal{M}_1 \cap \mathcal{M}_2\}$.

Many of the problems we have studied in this course may be formulated as matroid intersection problems.

Example 1: Find a maximum matching in a bipartite graph $G = (U \sqcup V, E)$.

Define

$\mathcal{M}_1 = \{A \subseteq E : \text{endpoints in } U \text{ of edges in } A \text{ are distinct}\}$

$\mathcal{M}_2 = \{A \subseteq E : \text{endpoints in } V \text{ of edges in } A \text{ are distinct}\}$

Observe that $\mathcal{M}_1$ satisfies the exchange property, since given $X, Y \in \mathcal{M}_1$ with $|X| > |Y|$ there are fewer vertices in U which are endpoints of edges in Y than there are vertices in U which are endpoints of edges in X. Let $x \in X$ be an edge of X whose endpoint in U is not the endpoint of an edge in Y. Notice, $Y \cup \{x\} \in \mathcal{M}_1$.

The set $\mathcal{M}_1 \cap \mathcal{M}_\in$ is the set of matchings in G.

Example 2: We show that Menger's problem may be formulated as a matroid intersection problem. We consider the following version:

Menger's Problem. Given an undirected graph $G = (V, E)$, and two subsets $S, T \subseteq V$ with $|S| = |T| = k$, we want to find k node-disjoint paths connecting S and T.

It is easy to reduce this problem to the case when G is connected. Let us construct two graphs G_1 and G_2 by contracting S and T to a single node, respectively. There is a natural bijection between E and the edge-sets of the two graphs G_1 and G_2. Hence the polygon matroids of of these graphs may be viewed as having the same underlying set E.

We claim that the largest common independent set of these two matroids a basis in common if and only if the answer to Menger's problem is affirmative. Suppose that G contains k node-disjoint paths from S to T. We can easily extend the union of these paths to a forest in G with k components, each containing exactly one of these paths. This forest is then a common basis in the two matroids. Conversely, a common basis of the two matroids is a forest with $n - k$ components, each of which meets both S and T in at most one node (else, the contraction of S or T would result in a circuit). So each of these components must meet both S and T in exactly one node, and hence contains a path from S to T. These k paths give a solution to Menger's problem.

These examples suggest that the alternating path method may be extendable to the matroid intersection problem. Such an algorithm for the unweighted case is the subject of our next section; we shall return to the weighted problem later.

7.6 A min-max theorem for matroid intersection

Given two matroids $(E, \mathcal{M}_1)$ and $(E, \mathcal{M}_2)$, in the restricted (unweighted) version of the problem we want to find $\max\{|A|, A \in \mathcal{M}_1 \cap \mathcal{M}_2\}$; the more general version also takes as input a weight function $c : E \rightarrow \mathcal{R}$ and asks for $\max\{c(A), A \in \mathcal{M}_1 \cap \mathcal{M}_2\}$.

We prove a min-max theorem for the unweighted case and sketch the polyhedral description and a polynomial time (ellipsoidal) algorithm for the weighted problem.

Theorem 7.3 *(Edmonds, Aigner-Dowling 1969)*

$$\max\{|A|, A \in \mathcal{M}_1 \cap \mathcal{M}_2\} = \min\{r_1(X) + r_2(E \setminus X), \quad X \subseteq E\} .$$

The rank function being the maximal size of an independent subset, the $\leq$ direction of the theorem is trivial: for every X and A, we have

$$\begin{array}{rcl} r_1(X) & \geq & r_1(X \cap A) \geq |X \cap A| \\ r_2(E \setminus X) & \geq & r_2((E \setminus X) \cap A) \geq |(E \setminus X) \cap A| \\ \text{sum} & \geq & |A|. \end{array}$$

The other inequality is proved by induction on the number of elements of the matroids. At each step we take an element e and either delete it or contract it.

If we delete e from both matroids, by induction there exists a set $A \subset E \setminus \{e\}$ in $\mathcal{M}_1 \cap \mathcal{M}_2$ and an $X \subset E \setminus \{e\}$ such that $|A| = r_1(X) + r_2(E - e - X)$. When we put e back in, we will have $|A| = r_1(Y) + r_2(E - Y)$ for either

$Y = X$ or $Y = X + e$, except in the bad case where $r_1(X) < r_1(X + e)$ and $r_2(E - X - e) < r_2(E - X)$; note that this bad case can't happen if either $r_1(e)$ or $r_2(e)$ is 0. When the bad case occurs, we use contraction instead of deletion.

If we contract e in both matroids, we get the new matroids defined by

$$B \in \mathcal{M}_i/e \Longleftrightarrow B \cup \{e\} \in \mathcal{M}_i \ .$$

(since e has rank 1 in both matroids). By induction there exists a B in the intersection and $Y \subset E - e$ such that $|B| = r'_1(Y) + r'_2(E - e - Y)$, where r'_i is the rank function of the contracted matroid. We know that

$$r'_i(Y) = r_i(Y \cup e) - 1.$$

Thus when we put e back in, taking the independent set $B + e$, for which we know that

$$|B + e| = r_1(Y + e) + r_2(E - Y) - 1 \ ,$$

we are done by taking Z equal either to Y or to $Y + e$, except in the bad case where $r_1(Y + e) = r_1(Y)$ and $r_2(E - Y) = r_2(E - Y - e)$.

When both bad cases occur for X and Y, we use the submodularity of the rank function. We consider the sum

$$\begin{aligned} S &= [r_1(X) + r_2(E - X - e)] + [r_1(Y + e) + r_2(E - Y)] \\ &= |A| + |B + e| + 1 \\ &\leq 2 \max |C| + 1. \end{aligned}$$

Using the submodularity of r_1 for the first and third terms, and the submodularity or r_2 for the other two, we get

$$\begin{aligned} S &\geq [r_1(X \cap Y) + r_1(X \cup Y \cup e)] + [r_2(E - X \cup Y \cup e) + r_2(E - X \cap Y)] \\ &\geq \max |C| + \max |C| \\ &\geq 2 \max |C| \end{aligned}$$

by putting the first and third terms together and the other two together. Since the difference between the lower bound and the upper bound on S is only 1, at least one of the two inequalities we used must be an equality, and thus either $X \cap Y$ or $X \cup Y \cup e$ must achieve equality, which concludes the proof of the theorem.

7.7 A matroid intersection algorithm.

We give a polynomial time algorithm to solve the unweighted matroid intersection problem. A similar, though more difficult, algorithm can be given for the weighted case; we will not discuss this here, but will return to give a polynomial time weighted matroid intersection algorithm as an application of linear programming methods in the last chapter.

Let $(E, \mathcal{M}_1)$ and $(E, \mathcal{M}_2)$ be two matroids. Assume that we have already found a common independent set A. We want to find a larger one or conclude that A is maximum.

Construct the following edge-colored bipartite graph on $V(G) = E$: for every element $v \in E \setminus A$, the set $A \cup v$ contains at most one circuit of $(E, \mathcal{M}_1)$. If it contains, then connect v to the other element of the circuit by red edges. Define the blue edges similarly, using the other matroid.

Let $S_i = \{e \in E : A \cup a \in \mathcal{M}_i\}$. If either S_i is empty, then A is trivially maximum. If S_1 and S_2 intersect, then any element in the intersection can be added to A. So we assume that S_1 and S_2 are disjoint non-empty sets.

Now do the following "alternating depth first search."

— Label the elements of S_1 by 0.

— Label by 1 their neighbors in the blue graph (these are all in A).

— Label by 2 the neighbors in the red graph of the points with label 1.

— Label by 3 the neighbors in the blue graph of the points with label 2, etc.

Let Y denote the set of point that get a label.

Case 1. $Y \cap S_2 = \emptyset$. Then by construction, every point $e \notin A \cup Y$ forms a circuit in $\mathcal{M}_1$ with some elements of A, and in fact this circuit is contained in $A \setminus Y$ (else, there would exist a red edge connecting e to $A \cap Y$, and so e would have been labelled). Hence $r_1(E \setminus Y) = r_2(A \setminus Y) = |A \setminus Y|$. Similarly $r_2(Y) = |A \cap Y$. Hence if $X = E \setminus Y$ then $|A| = r_1(X) + r_2(E \setminus X)$, which shows that A is maximum by Theorem XXX.

Case 2. $Y \cap S_2 \neq \emptyset$. Then there exists a path $b_0 a_1 b_1 \ldots a_k b_k$ such that the edges $b_{i-1} a_i$ are blue, the edges $a_i b_i$ are red, and $b_0 \in S_1$ and $b_k \in S_2$. By construction, there is no red edge connecting a_i to b_j if $i < j$ and no blue edge connecting a_i to b_j if $j < i - 1$.

We claim that the set $A' = A \setminus \{a_1, \ldots, a_k\} \cup \{b_0, \ldots, b_k\}$ is independent in both matroids. Let us give the argument for $(E, \mathcal{M}_1)$. We prove by induction on i that $A \setminus \{a_1, \ldots, a_i\} \cup \{b_0, \ldots, b_i\} \in \mathcal{M}_1$. This is trivial for $i = 0$ by the definition of S_1. Suppose we know it for an i and consider $i+1$. We know that $A \setminus \{a_1, \ldots, a_{i+1}\} \cup \{b_{i+1}\} \in \mathcal{M}_1$, since the only circuit that b_{i+1} forms with A contains a_{i+1}. So we can augment $A \setminus \{a_1, \ldots, a_{i+1}\} \cup \{b_{i+1}\}$ from $A \setminus \{a_1, \ldots, a_i\} \cup \{b_0, \ldots, b_i\}$ to an independent set of size $|A| + 1$. It suffices to show that we cannot pick up a_{i+1}, since then this augmented independent set must be $A \setminus \{a_1, \ldots, a_{i+1}\} \cup \{b_0, \ldots, b_{i+1}\}$. But if $A \setminus \{a_1, \ldots, a_i\} \cup \{b_{i+1}\}$ cannot be independent since by the remark about chords of the path, all elements of the circuit formed by b_{i+1} and A must lie in $A \setminus \{a_1, \ldots, a_i\} \cup \{b_{i+1}\}$.

This proves that we have found a larger common independent set.

We give a very brief survey of some results and methods in linear programming. This is not the main topic of the course; we shall, rather, concentrate on applications of linear programming techniques in combinatorial optimization.

A *linear program* is the problem of finding the maximum of a linear objective function

$$x \mapsto c^T x \tag{6}$$

(where $c \in \mathbb{R}^n$ is a fixed vector) subject to a finite number of linear inequalities. These inequalities can be summed up in matrix form as

$$Ax \le b, \tag{7}$$

where A is a matrix with m rows and m columns, and $B \in \mathbb{R}^m$.

The solution set of $Ax \le b$ is called the *feasible domain* of the program; any vector in the feasible domain is called a feasible vector (or a feasible solution). Any feasible vector maximizing the objective function is called an *optimum solution.*

There are many alternative ways to describe a linear program. We may want to maximize instead of minimize; we may have equations, and/or inequalities of the form $\ge$. Sometimes we consider only non-negative variables; the inequalities $x_i \ge 0$ may be included in (7), but it may be adventageous to separate them. All these versions are easily reduced to each other.

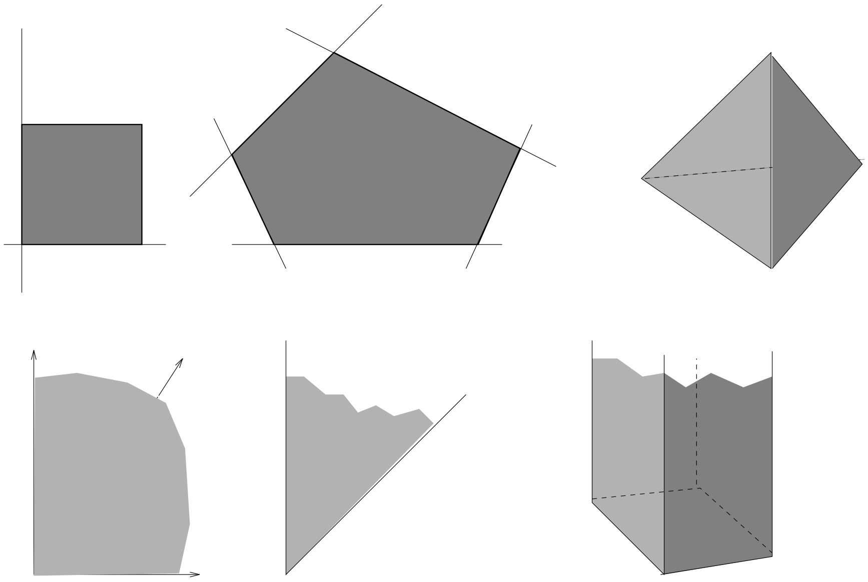

Figure 19: Polyhedra and polytopes

Often we are not interested in finding an optimum solution; rather, we would like to find just any feasible solution (or conclude that none exists). The general problem can in fact be reduced to this seemingly weaker problem. One way to do so is to consider the extended system $Ax \leq b$, $c^T x \geq \gamma$, and find by binary search the largest γ for which this system is solvable. (This of course gives an approximate optimum only; a better reduction uses the Duality Theorem below.)

7.8 Geometry of linear programs

A *polyhedron* is the solution set of a system of linear inequalities, i.e., a set of the form form $\{x \in \mathbb{R}^n : \ Ax \leq b\}$, where A is a matrix with m rows and n columns and b is a (column) vector of length m. A single linear inequality defines a *halfspace*, and so a polyhedron can be defined equivalently as the intersection of a finite number of hyperplanes.

For two points $x, y \in \mathbb{R}^n$, the *segment* connecting them is the set of all vectors $\lambda x + (1-\lambda)y$, $0 \leq \lambda \leq 1$. A set $X \subseteq \mathbb{R}^n$ is *convex* if for any two of its points x and y, the segment connecting x and y is contained in X. Given a set $S \subseteq \mathbb{R}^n$, its *convex hull* $\mathrm{conv}(S)$ is defined as the intersection of all convex subsets containing S. Clearly this is convex itself.

A *convex combination* of a finite set of vectors $v_1, \ldots, v_p \in \mathbb{R}^n$ is every vector of the form $\sum_{i=1}^p \lambda_i v_i : \lambda_i \geq 0, \sum_i \lambda_i = 1\}$.

Fact 7.2 *The convex hull of a set S is exactly the set of all convex combinations of finite subsets of S.*

Theorem 7.4 (Caratheodory's Theorem.) *Every vector in the convex hull of a set $S \subseteq \mathbb{R}^n$ is a convex combination of at most $n+1$ elements of S.*

A *polytope* is the convex hull of a finite number of points in $\mathbb{R}^n$.

Theorem 7.5 (Minkowski-Weyl Theorem.) *A set in $\mathbb{R}^n$ is a polytope if and only if it is a bounded polyhedron.*

This theorem provides a "good characterization" of membership in a polytope in the following sense. If we are given a polytope P as the convex hull of a list of vectors, then it is easy to certify that a vector v is in the polytope: all we have to do is to give a representation of v as a convex combination of the defining vectors. In fact, by Caratheodory's Theorem, we only have to use $n+1$ defining vectors. If P is given as the solution set of a system of linear inequalities, then it is trivial to certify that a point is not in P: all we have to show is a defining inequality that is violated.

Polyhedra may be unbounded; there is a way to describe unbounded polyhedra by appropriate versions of the convex hull construction, but we do not go into these details here. There is one class of unbounded polyhedra that is especially important. A *convex polyhedral cone* is the solution set of a system of homogeneous linear inequalities, i.e., a set of the form $\{x \in \mathbb{R}^n : \ Ax \geq 0\}$. A *conical combination* of a finite set of vectors $v_1, \ldots, v_p \in \mathbb{R}^n$ is every vector of the form $\sum_{i=1}^p \lambda_i v_i : \lambda_i \geq 0\}$. The *conical hull* of a set S is the set of all conical combinations of finite subsets of S.

Fact 7.3 *A set is a convex polyhedral cone iff it is the conical hull of a finite set of vectors.*

Let us mention some examples. The conical hull of the (standard) basis vectors $e_1, \ldots, e_n$ in $\mathbb{R}^n$ is the non-negative orthant $\mathbb{R}^n_+$, which can also be defined by the constraints $x_1 \geq 0$, …, $x_n \geq 0$. The conical hull of the set $\{e_1, -e_1\}$ is the line $x_1 = 0$. More generally, every linear subspace L is a convex polyhedral cone; if $b_1, \ldots b_k$ is a basis of L then L is the conical hull of $b_1, \ldots, b_k, -b_1, dots, -b_k$.

The (standard) cube can be defined as the solution set of the linear inequalities $0 \leq x_i \leq 1$, and also as the convex hull of all 0-1 vectors. The convex hull of the vectors $0, e_1, \ldots, e_n$ is called the *standard simplex*; this can also be defined by the linear inequalities $x_1 \geq 0$, …, $x_n \geq 0$, $x_1 + \ldots + x_n \leq 1$. The (standard) ball defined by the single quadratic inequality $\sum_i x_1^2 \leq 0$ is convex but not polyhedral: it cannot be defined by a finite set of linear inequalities.

Let P be a polyhedron. A *supporting hyperplane* for P is a hyperplane defined by a linear equation $a^T x = \alpha$ such that $a^T x \alpha$ for all points in P and $a^T x = \alpha$ for at least one point in P. The set of points on a supporting hyperplane is called a *face*. It is convenient to consider the polytope itself als a face. A single point forming a face is called a *vertex*. Any face of a polyhedron is a polyhedron, and a face of a face is also a face of the original polytope.

Fact 7.4 *Every polyhedron has a finite number of faces.*

Fact 7.5 *Let P be a polytope. Then every face of P is the convex hull of vertices contained in the face. In particular, P is the convex hull of its vertices. Furthermore, every set S whose convex hull is P contains all vertices.*

For unbounded polyhedra, a facet may not be determined by its vertices; a cone, for example, has at most one vertex. But the following is true:

Fact 7.6 *For every polyhedron P, the following are equivalent:*

(i) *P does not contain a full line;*

(ii) *P has a vertex;*

(iii) *Every face of P has at least one vertex.*

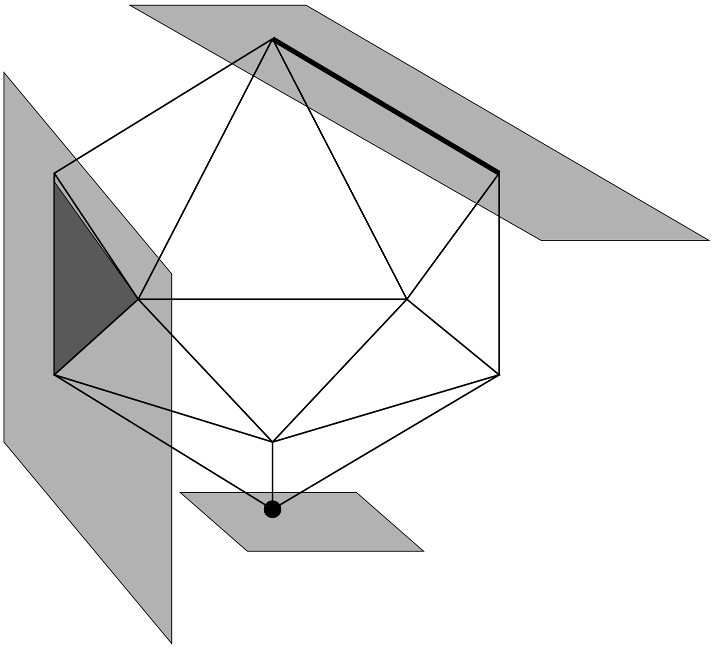

Figure 20: Faces of a polytope

An *affine subspace* of $\mathbb{R}^n$ is the solution set of a system of linear inequalities. Let $Ax = b$ define a non-empty affine subspace U, and let $x_0 \in U$. Then U can be obtained by translating the linear subspace V defined by $Ax = 0$ by the vector x. The *dimension* of U is the dimension of V (this is independent of the choice of x_0).

The *affine hull* of a set S is the intersection of all affine subspaces containing S. This is again an affine subspace.

The *dimension* $\dim(P)$ of a polyhedron P is the dimension of its affine hull. A vertex is just a face of P of dimension 0. An *edge* is a face with dimension 1. Every edge has two endpoints, which are vertices. Two vertices are *adjacent* (or *neighbors*) if the segment connecting them is an edge. A *facet* is a face with dimension $\dim(P) - 1$. A polyhedron is *full-dimnesional* if its dimension is n.

Fact 7.7 *Let P be a full-dimensional polyhedron. Every facet is defined (as a face) by a unique inequality (up to a positive scalar). The facet-defining inequalities have as their solution set the polyhedron itself. Every system of inequalities defining P contains all facet-defining inequalities (again, up to a positive scalar).*

Fact 7.8 *Intersection of facets is either empty or a face; conversely, every face (other than P itself) is the intersection of facets.*

Fact 7.9 *Let P be a full-dimensional polytope. Then every vertex of P is the intersection of all facet-defining supporting hyperplanes.*

In the description of the simplex method, it will be useful to know how a polytope looks like in the neighborhood of a vertex v. Let $a^T x \le b$ be an inequality valid for P such that v is the unique point in P with $a^T v = b$. So every other vertex u of P satisfies $a^T u < b$; since there is only a finite number of vertices, there is a $b' < b$ such that $a^T u < b'$ for every vertex $u \ne v$. The hyperplane $H = \{x \in \mathbb{R}^n : a^T x = b'\}$ intersects P in a polytope P'. The following is easy to show.

Fact 7.10 *Every k-dimensional face of P' is the intersection of H with a unique $(k+1)$-dimensional face of P containing v. In particular, the vertices of P' are the intersections of H with the edges of P incident with v.*

Let C_v be the cone spanned by the vectors pointing from v to its neighbors. It follows easily from the previous fact that

$$P \subseteq v + C_v = \{v + x : \ x \in C_v\}.$$

Fact 7.11 *Let v be a vertex of a polytope P defined by $Ax \le b$. Let A' be the submatrix formed by those rows of A for which the corresponding inequality is satisfied with equality by v, exists a matrix A' formed by certain rows of A, and let b' be the vector formed by the corresponding entries of b. Then v is the unique solution of the system $A'x = b'$.*

Theorem 7.6 *If P is a polytope defined by rational inequalities then every vertex of P is rational. If L is the number of bits in the description of A and b, then every vertex can be described using at most n^2L bits.*

(When describing a rational matrix or a vector, we write out every entry as a ratio of two integers in binary notation. The number 0 constributes one bit.)

This theorem implies, at least for the case when the set of feasible solutions is bounded, that the linear programming problem is well-posed in the sense that if A and b are rational with total input size L, and the problem of maximizing c^Tx subject to $Ax \le b$ has an optimum solution, then it has an optimum solution with a polynomial ($O(L^3)$) bits. This is also true for the unbounded case, and follows by imposing the additional constraints $-2^{nL} \le x_i \le 2^{nL}$).

7.9 Fourier–Motzkin elimination

Let

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots a_{1n}x_n &\le b_1 && (8)\\ a_{21}x_1 + a_{22}x_2 + \dots a_{2n}x_n &\le b_2 && (9)\\ &\vdots && (10)\\ a_{m1}x_1 + a_{m2}x_2 + \dots a_{mn}x_n &\le b_m && (11) \end{aligned}$$

be a system of linear inequalities. How do we know whether it is solvable? The simplest (but very inefficient) method for solving systems of linear inequalities goes back to Fourier in the last century; it generalizes the well known method of eliminating variables from linear equations. Note the difficulty: equations can be added and subtracted, and multiplied by any scalar (in the sense that this yields valid equations if we start with valid equations), but inequalities ("facing" in the same direction) can only be added and can only be mutiplied by non-negative scalars. So our possibilities for manipulating the system are much more restrected, and we pay a heavy price for this in complexity.

We can write each of the inequalities containing x_1 with non-zero coefficient either as

$$x_1 \le b'_j - a'_{j2}x_2 - a'_{j3}x_3 - \ldots - a'_{jn}x_n,$$

or as

$$x_1 \ge b'_j - a'_{j2}x_2 - a'_{j3}x_3 - \ldots - a'_{jn}x_n,$$

where $a'_{ji} = a_{ji}/a_{j1}$, $b'_j = b_j/a_{j1}$, and the direction of the inequality depends on the sign of a_{j1}. Assume, say, that the first k inequalities can be transformed into the first kind, the next r ineqaulities can be transformed into the second kind, while the rest does not contain x_1. We can write the system 8 as

$$x_1 \le \ell_1(x_2, \ldots, x_n) \tag{12}$$

$$\vdots \tag{13}$$

$$x_1 \le \ell_k(x_2, \ldots, x_n) \tag{14}$$

$$x_1 \ge \ell_{k+1}(x_2, \ldots, x_n) \tag{15}$$

$$\vdots \tag{16}$$

$$x_1 \ge \ell_{k+r}(x_2, \ldots, x_n) \tag{17}$$

$$0 \le \ell_{k+r+1}(x_2, \ldots, x_n) \tag{18}$$

$$\vdots \tag{19}$$

$$0 \le \ell_m(x_2, \ldots, x_n) \tag{20}$$

Now consider the following new system of linear inequalities, now only in $n-1$ variables:

$$\ell_{k+1}(x_1, \ldots, x_n) \le \ell_1(x_2, \ldots, x_n) \tag{21}$$

$$\ell_{k+2}(x_1, \ldots, x_n) \le \ell_1(x_2, \ldots, x_n) \tag{22}$$

$$\vdots \tag{23}$$
$$\ell_{k+r}(x_1,\ldots,x_n) \;\le\; \ell_1(x_2,\ldots,x_n) \tag{24}$$
$$\ell_{k+1}(x_1,\ldots,x_n) \;\le\; \ell_2(x_2,\ldots,x_n) \tag{25}$$
$$\vdots \tag{26}$$
$$\ell_{k+r}(x_1,\ldots,x_n) \;\le\; \ell_k(x_2,\ldots,x_n) \tag{27}$$
$$0 \;\le\; \ell_{k+r+1}(x_2,\ldots,x_n) \tag{28}$$
$$\vdots \tag{29}$$
$$0 \;\le\; \ell_m(x_2,\ldots,x_n) \tag{30}$$

Now it is easy to argue that 12 has a solution if and only if 21 has a solution; and in fact for every solution of 21 in $x_2,\ldots,x_n$ yeids an solution of 12 by choosing x_1 appropriately (the system 21 expresses exactly that no lower bound on x_1 contradicts any upper bound, and so we can choose x_1 arbitrarily in the range

$$\min\{\ell_j(x_1,\ldots,x_n):\ 1\le j\le k\} \ge x_1 \ge \max\{\ell_j(x_1,\ldots,x_n):\ k{+}1\le j\le k{+}r\}.$$

Now we can repeat this procedure and reduce the system to a system in $n-2$ variables etc. The algorithm terminates after n phases, when explicit numerical upper and lower bounds for x_n are obtained.

The main difficulty with this method is that the number of constraints grows terribly fast; starting with m constraints, we may have as many as $m^2/4$ constraints in the next step, so the increase is faster than exponential! This fact renders the Fourier-Motzkin elimination procedure practically useless. However, it does yield some important theoretical results; we discuss these in the next section, before going on to more efficient algorithms.

7.10 Farkas Lemma, Duality Theorem, and Complementary Slackness

The first lemma we prove gives a very useful necessary and sufficient condition for a system of linear inequalities to be solvable.

Lemma 7.3 *A system of linear inequalities* $a_1^T x \le b_1,\ \ldots,\ a_m^T x \le b_m$ *has no solution iff there exist* $\lambda_1,\ldots,\lambda_m \ge 0$ *such that that* $\sum_i \lambda_i a_i = 0$ *and* $\sum_i \lambda_i b_i = -1$.

Proof. The "if" part is obvious. Conversely, if the system has no solution, then the Fourier-Motzkin elimination algorithm produces, at termination, an inequality $0 \leq d$ with $d < 0$. Every inequality produced during the Fourier Motzkin elimination is a non-negative linear combination of the original inequalities. In particular, we have non-negative multipliers $\lambda'_1, \ldots, \lambda'_m$ to produce $0 \leq d$; these multipliers give $\sum_i \lambda'_i a_i = 0$ and $\sum_i \lambda'_i b_i = d$. Dividing by $-d$, we get the multipliers in the lemma. ∎

Lemma 7.4 (Farkas Lemma, homogeneous version) *Let $a_1, \ldots, a_m, c \in \mathbb{R}^n$. Then $c^T x \geq 0$ for all solutions of $a_1^T x \geq 0, \ldots, a_m^T x \geq 0$ iff c is in the cone generated by the a_i, i.e., there exist $\lambda_1, \ldots, \lambda_m \geq 0$ such that $c = \sum_i \lambda_i a_i$.*

Proof. (First proof) The "if" part is obvious. Conversely, if $c^T x \geq 0$ for all solutions of $a_1^T x \geq 0, \ldots, a_m^T x \geq 0$, then the system $a_1^T x \geq 0, \ldots, a_m^T x \geq 0, c^T x \leq -1$ has no solution. Applying Lemma 7.3, we get the λ_i as required. ∎

Proof. (Second proof; sketch) Assume that c is not in the cone K generated by the a_i. Then there exists a hyperplane $u^T y = u_0$ separating c from the cone; say, $u^T c < u_0$ but $u^T y \geq u_0$ for all $y \in K$. Since the origin is in the cone, we have $u_0 \leq 0$. Since together with any $y \in K$, we also have $\mu y \in K$ for an arbitrarily large μ, it follows that $u^T y \geq 0$ for all $y \in K$. In particular, $u^T a_i \geq 0$ for all i. So $x = u$ violates the hypothesis. ∎

Lemma 7.5 (Farkas Lemma, inhomogeneous version) *Let $a_1, \ldots, a_m, c \in \mathbb{R}^n$, $b_1, \ldots, b_m, d \in \mathbb{R}$. Assume that the system $a_1^T x \leq b_1, \ldots, a_m^T x \leq b_m$ has a solution. Then $c^T x \leq d$ for all solutions of $a_1^T x \leq b_1, \ldots, a_m^T x \leq b_m$ iff there exist $\lambda_1, \ldots, \lambda_m \geq 0$ such that $c = \sum_i \lambda_i a_i$ and $d \geq \sum_i \lambda_i b_i$.*

Proof. The "if" part is obvious. Conversely, assume that $c^T x \leq d$ holds for all solutions of $a_1^T x \leq b_1, \ldots, a_m^T x \leq b_m$. Let z be a new variable and consider the system

$$b_1 z - a_1^T x \geq 0, \ \ldots, \ b_m z - a_m^T x \geq 0, \ z \geq 0.$$

We claim that every solution of this satisfies the inequality $dz - c^T x \geq 0$. If $z > 0$ then this is trivial by dividing through by z. Assume that the system has a solution (z, x_1) with $z = 0$ and $dz - c^T x_1 = -c^T x_1 < 0$. Let x_0 be a solution of the original system. Then for every $N > 0$, the vector $x_0 + N x_1$ is a solution of the original system $a_1^T x \leq b_1, \ \ldots, \ a_m^T x \leq b_m$, but for large N, $c^T(x_0 + N x_1) > d$, a contradiction.

So we can apply Lemma 7.4, and the lemma follows.

Let

$$\begin{align} \text{maximize} \quad & c^T x \tag{31} \\ \text{subject to} \quad & Ax \leq b. \tag{32} \end{align}$$

be a linear program (called the *primal program*). Its *dual* is the linear program

$$\begin{align} \text{minimize} \quad & b^T y \tag{33} \\ \text{subject to} \quad & A^T y = c, \quad y \geq 0. \tag{34} \end{align}$$

One can define more generally (and more symmetrically) the dual of a linear program of the following more general form:

$$\begin{align} \text{maximize} \quad & c^T x \tag{35} \\ \text{subject to} \quad & A_1 x = b_1 \tag{36} \\ & A_2 x \leq b_2 \tag{37} \\ & x_1 \text{ unrestricted} \tag{38} \\ & x_2 \geq 0. \tag{39} \end{align}$$

(here $\binom{A=(A_1}{A_2)}$ is a splitting of the rows of A into two groups, and $\binom{x=(x_1}{x_2)}$ is a splitting of the entries of x into two groups. To obtain the dual, we consider the splitting of the columns of A into two groups (A^1, A^2) corresponding to the splitting of x, we consider a variable for each row of A, which are then split into two groups according to the splitting of the rows of A, to get two vectors y_1, y_2. The dual program is then

$$\begin{align} \text{minimize} \quad & b^T y \tag{40} \\ \text{subject to} \quad & A^1 y = c_1 \tag{41} \\ & A^2 y \geq c_2 \tag{42} \\ & y_1 \text{ unrestricted} \tag{43} \\ & y_2 \geq 0. \tag{44} \end{align}$$

So every primal constraint (not counting non-negativity) corresponds to a dual variable and vice versa. Equality constraints correspond to unrestricted variables, inequalities correspond to non-negative variables. The coefficient of variable i in constraint j of the primal program is the same as the coefficient of variable j in constraint i in the dual.

We could further complicate matters by considering inequalities with both $\geq$ and $\leq$ in both programs, and non-positive as well as non-negative variables. All these generalizations. however, are easily reduced to the special form (31) above, and for simplicity of presentation we shall restrict ourselves to that.

The central result in linear programming is the following theorem.

Theorem 7.7 (Duality Theorem) *If either one of the primal and dual programs has an optimum solution, then so does the other and the two optimum values are equal.*

We may add the following.

Supplement. *The primal program is infeasible iff the dual is unbounded. The dual program is infeasible iff the primal is unbounded.*

Proof. I. Let x be any solution of the primal program and y, any solution of the dual. Then, using the inequalities $y \geq 0$ and $Ax \leq b$, we get

$$c^T x = (A^T y)^T x = y^T Ax \leq y^T b = b^T y.$$

Hence the maximum of $c^T x$ is at most the minimum of $b^T y$. (This inequality is called *weak duality*)

II. Assume that e.g. the primal has an optimum solution, and let γ be the optimum value. Then $c^T x \leq \gamma$ holds for all solutions of $Ax \leq b$, and hence by Lemma 7.5, it can be derived by taking non-negative linear combinations. This means formally that we have a vector $y \geq 0$ such that $A^T y = c$ and $b^T y \leq \gamma$. Since $b^T y \geq \gamma$ by the first part of the proof, we have equality.

III. Assume that the dual problem is unbounded. Then by the weak duality, the primal cannot have a feasible solution.

Assume that the primal problem has no feasible solution. Then by lemma 7.3, there exists a vector $y \geq 0$ such that $A^T y = 0$ and $b^T y = -1$. Since we can multiply y by any positive constant, this implies that the dual is unbounded. ∎

Looking at the first part of the previous proof (weak duality), we can read off the following (again, we leave the formulation for the most general dual pair or programs to the reader).

Theorem 7.8 (Complementary slackness) *Let x be a solution of the primal program and y, a solution of the dual program. The both x and y are optimal if and only if for every j with $y_j > 0$, the jth constraint of the primal problem is satisfied with equality.* ∎

One consequence of the Duality Theorem is that the problem of maximizing a linear objective function $c^T x$ subject to a system of linear inequalities $Ax \leq b$ can be reduced to the problem of finding a feasible solution. In fact, consider the following system of linear inequalities (and equations):

$$Ax \leq b, \quad A^T y = c, \quad y \geq 0, \quad c^T x = b^T y$$

Every solution of this gives an optimum solution of the original linear program.

7.11 The Simplex Method

The method most often used to solve linear programs in practice is the simplex method. This was developed by Dantzig in 1947.

We start with a geometric description of the idea. Let $Ax \leq b$ be a system of linear inequalities and assume, for the sake of simplicity of presentation, that its solution set is a polytope P. We want to maximize a linear objective function $c^T x$. We know that the maximum is attained at at least one vertex. So let us try to find a vertex that maximizes the objective function.

We start with finding an arbitrary vertex v_0 (we'll come back to the question of how to do so), and then, starting from this vertex, walk along the edges of P from vertex to vertex so that the objective function increases at each move. If we get stuck, then we are at a vertex v with the property that no neighboring vertex has better objective function value. The following lemma shows that such a vertex is optimal.

Lemma 7.6 *If u is a vertex such that no neighbor of u has a higher objective function value than u then u is an optimal vertex.*

Proof. . We know that $P \subseteq u + C_u$, where C_u is the cone spanned by the vectors $v - u$, where v is a neighbor of u. By hypothesis, $c^T(v - u) < 0$ for each of these vectors. Hence $c^T x < 0$ for all $x \in C_u$, and so $c^T(x - u) < 0$ for all $x \in P$. ∎

(The previous lemma implies that the graph formed by the edges and vertices of a polytope is connected. But how large is the diameter of such a graph, given n, the dimension and m, the number of facets. This question is very improtant from the linear programming point of view since this diameter gives a lower bound on the number of steps in the simplex algorithm. The **Hirsch conjecture** states that for a polytope, this diameter is at most $m - n$. However, no polynomial upper bound for this diameter is known. A recent result of Gil Kalai is that the diameter is bounded by $m^{\log m^{const}}$.)

This very informal description needs a lot of patching up. The most important issue is: how to find an adjacent vertex with better objective function value? If we write out this problem formally, it looks like an $(n-1)$-dimensional linear programming problem, and so this seems like a very bad recurrence.

Assume first that the polytope is *simple*, i.e., every vertex is contained in exactly n facets. In this case, neighbors of a vertex u are exactly those vertices that share $n - 1$ of these facets. So there are exactly n neighbors, and they are easily found by choosing $n - 1$ of the contraints that are tight at u, choosing one further constraint, and then solving the resulting system of linear equations. Let u' be the resulting vector; then we have to check two things about u': first, that it is feasible, and second, that the objective function value is better. (It is clear that this involves testing at most $O(nm)$ points u'; in practice, much more economical ways are known.)

The case when the polytope is not simple represents new difficulties. In this case, many constraints may be tight at the current vertex and we may have exponentially many edges going out. One remedy is to "perturb" the system, by adding tiny constants to the right hand sides so that the polytope becomes simple but the optimal solution does not change much.

A more elegant version of this idea is to add "infinitesimally small" quantities to the right hand sides. This means that we don't have do to them anything numerical, but, say, imagine that we have added ε^k to b_k, and allow moving on the "infinitesimally short" edges that arise.

But it is better to give up the picture of moving from vertex to vertex, and move from n-tuples of defining inequailities to n-tuples of defining inequalities. We call an n-tuple a *basis* if the right hand sides are linearly independent, and a *feasible basis* if the unique common point of the corresponding hyperplanes is a feasible solution to the whole system. This common point is necessarily a vertex of P, but the same vertex may correspond to many feasible bases. A simplex step (or *pivot step*) consists of removing an element from the basis and replacing it with another one such that we get another one so that we get a new feasible basis, and the objective value for the vertex determined by this new objective value is not worse than the previous.

Unfortunately, we cannot require any more that the objective value strictly increases at each step. We have to allow *degenerate steps*, i.e., steps where the new basis determines the same vertex. On the other hand, to allow any such step may lead to cycling, and so an important question is to find simple restrictions on simplex steps which prevent cycling. A rule that tells which constraint to leave out from the basis and which constraint to add is called a *pivot rule.*

One such rule can be derived from the "infinitely small perturbation" discussed above; this leads to the so-called *lexicographic anticycling rule.* Several other simple rules are known to prevent cycling.

The issue of pivot rules goes, however, beyond anticycling. The running time of the simplex method depends on this rule. It is not known if any pivot rule exists that makes the simplex method polynomial time.

How to find a starting vertex? One solution is to consider the inequalities one by one. Any n of them gives a vertex. If we have a vertex v that satisfies the first k, then consider the $k+1$-st constraint $a^T x \leq \alpha$. If this is not satisfied, minimize the objective function $a^T x$ subject to the previous constraints, to get a new vertex v'. Now if there is any feasible vertex, then v' will satisfy $a^T x \leq \alpha$.

There are a lot of subtleties in implementing the above simple geometric considerations, leading to various versions of the Simplex Method. We refer to (Papadimitriou and Steiglitz 1982) and (Chvàtal 1983) for details.

7.12 The Ellipsoid Method

We now give an outline of a method for the general linear programming problem, known as the *ellipsoid method.* The method was developed by Khachiyan (1979), based on previous results of Shor and Yudin–Nemirovskii in non-linear optimization. This was the first polynomial-time algorithm to solve linear programs.

Given a convex polytope $K \subset \mathbb{R}^n$, the solution set of a linear program 8, and a non-zero vector $c \in \mathbb{R}^n$, we want to compute $\max\{c^T x \mid x \in K\}$.

As a preparation, we have to introduce some notions from geometry. A matrix is *positive definite* if it is symmetric and all of its eigenvalues are positive. An *ellipsoid* with center $v_0 \in \mathbb{R}^n$ is defined by $E = \{x \mid (x - v_0)^T A(x - v_0) \leq 1\}$, where A is an $n \times n$ positive definite matrix. Equivalently, an ellispoid could be defined as the image of a ball under a linear transformation.

First we describe the ellisoid algorithm for the problem of finding a solution to the system of inequlities 8, i.e., if finding a point in K.

At a given stage of the algorithm, we shall have an ellipsoid E containing K. To be able to start, we assume that we know some number $R > 0$ such that the ball $B(0, R)$ with radius R and center 0 contains k; (It is usually easy to find such a ball. In fact, in most combinatorial examples K is a polyhedron with vertices in $\{0, 1\}^n$, and so K is contained in $B(0, \sqrt{n})$.)

Let v_0 be the center of E. If $v_0 \in K$ we are done; so suppose that $v_0 \notin K$. Then v_0 violates one of the constraints defining K, i.e., one of the hyperplanes $\sum_i a_{ji} x_i \leq b_j$ does not contain v_0. Let $\hat{E}$ denote the part of the ellipsoid E that does satisfy this constraint; this is less than half of the ellipsoid. It is then possible to define an ellipsoid E' which contains $\hat{E}$ (and therefore K), whose volume satisfies

$$\mathrm{vol}(E') < e^{-1/3n} \mathrm{vol}(E).$$

Then we replace E by E', and repeat. (see figure 21 for an illustration.

We note that it is not clear that the successive centers of the ellipsoids ever fall inside K. For example, if K is a single point with transcendental coordinates, the centers of the ellipsoids all have algebraic coordinates

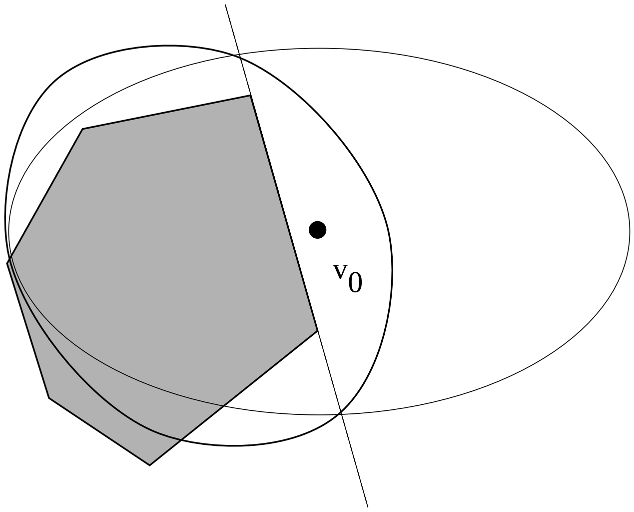

Figure 21: The ellipsoid method

and so are never in K. However the situation is different if K is full-dimensional. We in fact need full-dimensionality in the following effective sense: assume that we know a positive ϵ such that $\mathrm{vol}(K) \geq \epsilon$. (Note that the assumption that we know a ball containing K may be viewed in a similar spirit as an effective version of the assumption that K is bounded.)

Claim: *The ellipsoid algorithm finds a point inside K in polynomial time.*

In fact, as long as the ellipsoid centers are outside K, at every iteration the algorithm computes a new ellipsoid, and the volumes of these ellipsoids decrease exponentially; but since they all contain K, they have to be greater than ϵ. If $E_0, E_1, \ldots$ are the successive ellipsoids, we have

$$\begin{aligned} \epsilon &\leq \mathrm{vol}(E_m) \\ &\leq e^{-m/3n}\mathrm{vol}(E_0) \\ &\leq e^{-m/3n}(2R)^n \end{aligned}$$

from which we conclude that after we reach $m = 3n(\log\frac{1}{\epsilon} + n\log(2R))$, we have to have one of the ellipsoid centers x_i inside K. The number of iterations is thus polynomial in the number of bits of ϵ and R.

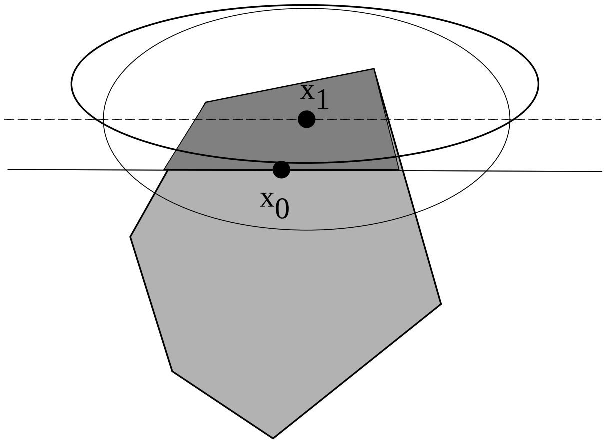

Figure 22: The sliding objective function

If we want not just a feasible point, but a point maximizing (say) the objective function $c^T x$, then one possiblity is to try to find a point in $K_\gamma := \{x \in K : c^T x \geq \gamma\}$ for various values of γ. (It is trivial to design a separation routine for this body.) If the ellipsoid algorithm runs too long, i.e., the ellipsoids shrink to a very small size, then we know that K_γ is (almost) empty; otherwise, we find a solution with objective value at least γ. We can do binary search in γ to find a good approximation of the optimum solution.

A more elegant alternative is the method of *sliding objective function.* At any stage, we have a value γ, which is the best objective value found so far, the corresponding best solution x_0, and an ellipsoid E with center x_1 that includes K_γ (we start with $\gamma = -\infty$ and $E = B(0, R)$).

Three cases now occur.

Case 1: $x_1 \in K$ and $c^T x_1 > \gamma$. Then we can replace x_0 by x_1, γ by $c^T x_1$, and keep the same ellipsoid E.

Case 2: $x_1 \in K$ but $c^T x_1 \leq \gamma$. We consider the hyperplane $c^T x = \gamma$. This hyperplane goes through the center of the ellipsoid, and K' is all on one side of it (figure 22). **Case 3**: $x_1 \notin K$. Then we proceed as in the algorithm for finding a feasible point.

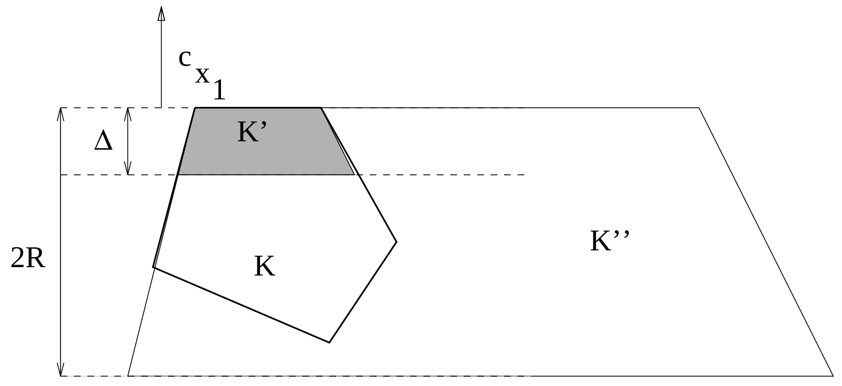

Figure 23: Near the optimum solution

Note that whenever case 1 arises then it is followed by an occurence of case 2. Thus the number of steps is at most twice the number of ellipsoids defined. Thus the algorithm provides a better and better approximation of the region containing the optimal solution using smaller and smaller ellipsoids.

As described so far, we have an infinite procedure. We know from the "feasibility" version that aftyer a polynomial number of steps, we find a point in the body. But even if we are inside the convex body, how close do we get to the optimal solution?

Assume that we do not get closer than Δ from the optimal solution (see figure 23). Let x_1 be an optimal solution and let K' be the set of points of K within distance Δ from the hyperplane $c^T x = c^T x_1$. If we blow up K' by a factor of $2R/\Delta$, using a homothetical transformation with center x_1, defined by

$$x \to (x - x_1)\frac{2R}{\Delta} + x_1$$

(where x_1 is one of the optimal solutions), then the new body K'' contains K. This is due to the convexity of K and to the fact that the diameter of K is at most $2R$ (so that no point of K can be at distance more than $2R$

from x_1). Thus

$$(\frac{2R}{\Delta})^n \mathrm{vol}(K') = \mathrm{vol}(K'') \geq \mathrm{vol}(K) \geq \epsilon \ .$$

Thus $\mathrm{vol}(K') \geq \epsilon(\Delta/2R)^n$, and one of the ellipsoid centers will eventually have to be inside K', using the same argument as before, after at most

$$3n(\log\frac{1}{\epsilon} + 2n\log(2R) + n\log(\frac{1}{\Delta}))$$

stages. This center x_i is in K and satisfies $c^T x_i \geq \max_K\{c^T x\} - \Delta|c|$.

Thus we have proved the following result:

Theorem 7.9 *Let $K \in \mathbb{R}^n$ be a closed convex set and $c \in \mathcal{Q}^n$. Assume that we have a separation oracle for K. Also assume that we know positive real numbers R and ϵ such that $K \subseteq B(0, R)$ and vol$(K) \geq \epsilon$. Then with the ellipsoid method, in time polynomial in n, $\log\epsilon$, $\log R$, and $\log\Delta$ (where the time is measured as the number of steps), we get a solution $x_0 \in K$ such that*

$$c^T x_0 \geq \max\{c^T x, \quad x \in K\} - \Delta|c| \ .$$

We have skipped a number of technical details in this presentation: namely, how to define a new, smaller ellipsoid (this is fairly basic geometry), and how to deal with numerical problems: the entries of the matrix defining the ellipsoid must not be too large if we want polynomial time; the rounding must be precise. However these are all technical details, and we do not have enough time to discuss them. (See L. Lovász, An Algorithmic Theory of Numbers, Graphs and Convexity, CBMS–NSF 50, SIAM 1986 for some details and M. Grötschel, L. Lovász and A. Schrijver, Geometric Algorithms and Combinatorial Optimization for full details.)

7.13 Polyhedra with 0/1 vertices.

The methods described in the previous part only give an approximately optimal solution. In this generality, we cannot expect to get the real optimum (it may be irrational, for example). But we can get the exact optimum if we consider polyhedra with not too weird rational vertices.

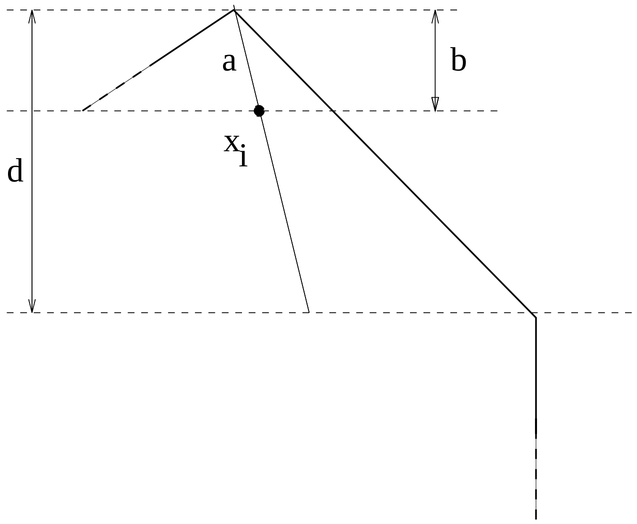

Figure 24: Rounding the coordinates.

Assumption 3: We shall restrict ourselves to polyhedra with vertices in $\{0,1\}^n$. (For combinatorial applications, these are the most important.)

Let us first suppose that the optimal solution is unique. Assume that the objective function c has rational coordinates, and that their least common denominator is q. Then the value of c at any vertex is an integer times $1/q$. Thus the minimal "vertical" distance d between the optimal solution x^* and the other vertices of K (vertical means the direction of c) satisfies $d \geq |c|/q$. Hence it appears that once the ellipsoid method finds a point close enough to the optimal solution, simply rounding the values of the point will give the optimal solution x^*. How close should we get to insure that we will not get the wrong point?

A simple geometric argument gives a bound on the number of iterations needed (See figure 24). Let a be the distance between x_i and x^*. If $a < 1/2$ we are guaranteed that rounding the coordinates of x_i can give no other point than x^*. Let b be the vertical distance between x_i and x^*. If we follow the line (x^*, x_i) until we get at vertical distance d from x^*, since all the other vertices of K are below that we are still in K; so the distance must be at most $2R$. In other words,

$$a \leq 2R\frac{b}{d} \ .$$

We just need to choose Δ so that $2R\Delta/d < 1/2$ to insure that any point in K_Δ gives x^* by rounding the coordinates. Thus we choose

$$\Delta < \frac{1}{4q|c|R} \ ,$$

and run the algorithm long enough to insure that we get a point inside K_Δ. Rounding the coordinates of that point will gives the exact optimum solution.

Some of the assumptions we made were not really essential. For example, we assumed that the optimum solution was unique: actually, we can always insure that by doing a slight pertubation of c.

Claim: If we replace c by $c_1 = c + (1/N, 1/N^2, \ldots, 1/N^n)^T$, with N big enough, the optimum solution becomes unique.

In fact, assume that x and x' are distinct optimum solutions. Then $c_1x = c_1x'$ and so $c(x-x') = c'(x-x')$ (where $c' = c - c_1$). Two cases need to be considered.

Case 1: $c(x-x') \neq 0$. Then since the vertices have integer coordinates, we have $c(x-x') > 1/q$. Choosing $N > 2q$ or so guarantees that $c'(x-x') < 1/q$: contradiction.

Case 2: $c(x-x') = 0$. Then we choose N larger than the largest coordinate of c, and since $1/N >> 1/N^2 >> \ldots$, the first non-zero term dominates the sum, and $c'(x-x') \neq 0$: contradiction.

Another assumption we made was to suppose that $\mathrm{vol}(K) > \epsilon$. But if K is full-dimensional with 0-1 vertices, then it can be decomposed into simplices and each of them has volume at least $1/n!$, so $\mathrm{vol}(K) \geq 1/n!$.

Summary: If K is full-dimensional, with 0-1 vertices, and there is a separation oracle for K, then we can take $\epsilon = 1/n!$ and $R = \sqrt{n}$, and every linear function can be optimized over K in linear time.

We note that the assumption that K is full-dimensional can be removed, though at the cost of some hard non-trivial work.

The issue of how the separation oracle works is the essential question. This must be designed separately for each application. We shall see that this is often possible.

Let us state the more general form of the theorem.

Theorem 7.10 *Let K be a polyhedron in $\mathbb{R}^n$. Suppose that we know $\phi > 0$ such that K can be defined by inequalities of the form $\sum a_i x_i \leq \alpha$, where $|a_i| \leq 2^\phi$, $|\alpha| \leq 2^\phi$, and a_i and α are integers. Assume that we have a separation oracle for K. Then* $\max\{c^T x, x \in K\}$ *can be found in time poly(ϕ, n, #digits in c).*

Note that K can be defined by an arbitrary number of inequalities. In this general form, K does not have to be full-dimensional, and does not have to be bounded. Note that one condition for the algorithm to be polynomial-time is that the separation oracle must be "good", that is, the vector a defining the separating hyperplane must not have exponentially many digits: we should have #digits of a =poly(#digits in y).

What is the value of ϕ if K has 0-1 vertices? Then each facet contains n 0-1 vertices $v_1, v_2, \ldots v_n$. Its equation is of the form $a^T x = \alpha$. Thus we have the system of equations

$$\begin{cases} a^T v_1 & = \alpha \\ a^T v_2 & = \alpha \\ \ldots & \\ a^T v_n & = \alpha \end{cases}$$

which we can solve in (a, α) (up to a constant factor). Choosing the factor adequately to avoid denominators, we find that $a_i(-1)^{i-1}$ is the subdeterminant obtained by dropping the i^{th} column in the following matrix

$$\begin{pmatrix} - & v_1 & - & (-1) \\ - & v_2 & - & (-1) \\ & \ldots & & \\ - & v_n & - & (-1) \end{pmatrix}$$

Since all the entries of the matrix are 0,1 or -1, we find that $|a_i| \leq n!$ so that $\phi = n \log n$ works.

Conversely, if for all i we have $|a_i| \leq 2^\phi$ and $\alpha \leq 2^\phi$, then the coordinates of the vertices, which are solution of a system of linear equations in a_i and α, cannot be too "ugly".

The main issue is how to find the separation oracle. Clearly that is an easy task if K is defined by explicitly given inequalities, of the form $K = \{x, Ax \leq b\}$; then, given y, the oracle can just compute Ay. If it is $\leq b$, then $y \in K$; otherwise the constraint that y does not satisfy can be used as a separator. In particular, this implies that linear programming can be solved in polynomial time.

7.14 The Barrier Method

This method was developed by Karmarkar (1984). As opposed to the simplex method, which steps from vertex to vertex, and the ellipsoid method, where the centers of ellipsoids approach the optimum solution jumping in and out from the feasible domain, Karmarkar's method generates a sequence of feasible vectors converging to the optimum. We describe some of the geometric background.

Assume that we want to maximize a linear objective function $c^T x$ over a convex body K. Also assume that we have already found an interior point x_0 and we want to find a sequence of interior points $x_1, x_2, \ldots$ converging to the optimum point. When stepping from x_i to x_{i+1}, it would be natural to move in the direction of c, or, in other words, orthogonal to the hyperplane $c \cdot x = \text{const}$. This leads us, however, to points close to the boundary and (possibly) far from the optimum, and then we have to move in some other direction.

However, it is not really natural to move orthogonal to c; this direction is not invariant under affine transformations while the hole task is. The idea of many optimization algorithms is to change the metric of the space so that the notion of "orthogonality" should be tailored to the particular body (such are the variable metric methods, but the ellipsoid method can also be viewed this way).

There is a very natural metric inside every convex body, introduced by Hilbert. The distance of any two points $u, v \in \text{int}(K)$ is defined as the logarithm of the cross ratio of u, v, and the two points of intersection of the line through u and v with the boundary of K. (This construction may also be familiar from the Caley–Klein model of hyperbolic geometries.) It is not difficult to see that if we move in the direction of steepest descent with respect to this metric to, say, half way to the boundary, and then repeat this, then we get very close to the optimum very fast. For example, if K is centrally symmetric with respect to x_0 then we approach the optimum on a straight line and the distance from the optimum is halved at each step.

Unfortunately, there is no easy way to compute the Hilbert metric in a convex body, but we can approximate it in the neighborhood of an interior point v by finding a projective transformation which maps K onto a convex body K' so that the unit ball includes K, K includes the ball with a reasonably large radius ρ about the origin, and v is mapped onto the origin. Then the euclidean metric will approximate the projective metric. Except for the condition on v, this sounds like the ellipsoid method, but we have more freedom because we are allowed to use projective transformations. (The optimization problem itself is not projective invariant, but one can transform it into projective invariant problems easily; for example, the Violation Problem above is projective invariant.)

The crucial part of Karmarkar's method is to construct such a projective transformation very fast in the case of polytopes presented by an explicit system of m linear inequalities. The inner radius ρ he achieves is $1/m$; this shows that (unlike the ellipsoid method) his method is sensitive to the number of constraints. This is of course only a very rough sketch; many ingenius details must be added to make this procedure work for really large linear programs. For details, extensions, and related algorithms see the survey of Todd (1989).

7.15 Complexity Considerations

We shall consider only the *solvability problem of linear inequalities*:

Problem. Given a rational $m \times n$ matrix A and a rational vector b, is the system of linear inequalities $Ax \leq b$ solvable?

This problem is in NP (just exhibit a solution; one has to show though that if there is a solution at all, then there is one with small rational entries).

the problem is also in co-NP (this follows by the Farkas Lemma 7.3).

We have discussed four algorithms to solve this problem. The first, Fourier–Motzkin elimination, is exponential (and useless in practice). Its main use was to derive the Farkas Lemma.

The second, the simplex method, is also exponential in the worst case (at least for most known pivot rules; it is open whether any pivot rule would make it polynomial). However, it is very efficient in practice.

The ellipsoid method is polynomial in the worst case, but too slow for almost all practical applications. However, it is extremely important from the theoretical point of view. Not only does it show that the solvability of linear inequalities is in P, but it has the important feature that it is not necessary to explicitly list all constraints; it suffices to have a subroutine (oracle) that generates a violated constraint for a given infeasible point.

The barrier method is also polynomial in the worst case, and in addition it performs very well in practice (it is competitive with the simplex method). It does not seem to have all the theoretical adventages of the ellipsoid method, however.

It should be remarked that the simplex method has an additional theoretical feature: the number of arithmetic operations it performs can be bounded by n and m (independently of the magnitude of the entries of A and b). Unfortunately, this bound is exponential (at least for the known versions). The number of arithmetic operations performed by the ellipsoid and barrier methods, however, depends on the entries. It is an outstanding open problem whether there is a linear programming algorithm that is *strongly polynomial*, i.e., which performs $O((nm)^{const})$ arithmetic operations (on numbers whose bit-size is polynomially bounded in L, the number of bits in the input).

8 Polyhedral combinatorics

8.1 The matching polytope

8.1.1 Minimum weight perfect matching in a bipartite graph

Given a graph $G = (V, E)$ that has a perfect matching, and a cost function $w : E \to \mathcal{R}$, we want to find a perfect matching M of minimum weight, where the weight of a matching is defined by $w(M) = \sum_{e \in M} w(e)$.

This problem is polynomial time solvable in general, but first we deal with the case when G is a bipartite graph with bipartition (A, B), where $|A| = |B|$. We may restrict our attention to the case shen G is a complete bipartite graph, since missing adges can be added with weight ∞. We may also assume that all the weights are non-negative. Let G_0 denote the graph formed by edges with weight 0. Using the unweighted bipartite matching algorithm, we can test whether G_0 has a perfect matching. If this the case, we are done, since this perfect matching is clearly optimal. Otherwise, we adjust the weights so as to change the set of 0-edges in a suitable way.

If there is a vertex v such that all the edges adjacent to it have positive weight, we can subtract a constant from all those edges: since any perfect matching must use exactly one edge adjacent to v, that operation increases the number of 0-edges and does not change the minimum perfect matching. However this transformation is not always sufficient to find a perfect matching with all weights 0., as can be seen from the example figure 25. But we know that in case of failure, the alternating path algorithm for finding a 0-weight perfect matching finds a set $X \subseteq A$ with $|Y| < |X|$, where Y is the set of neighbors of X in G_0. We want to increase the number of 0-edges out of X. If ϵ is the minimum weight of an edge from X to $B \setminus Y$, adding ϵ to all the edges out of Y and $-\epsilon$ to all the edges out of X will keep the values of the edges between X and Y the same, while increasing the number nodes connected to X by 0-edges. See figure 26. Through this transformation, any perfect matching changes its weight by $-\epsilon|X| + \epsilon|Y|$, which does not depend on the matching, so the same matching(s) will be optimal.

It remains to show that this procedure terminates, and to estimate the number of iterations it takes. One way to show this is to remark that the set X

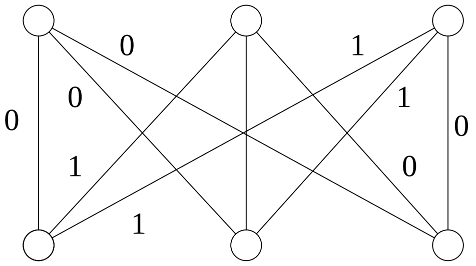

Figure 25: Local subtraction is not always sufficient

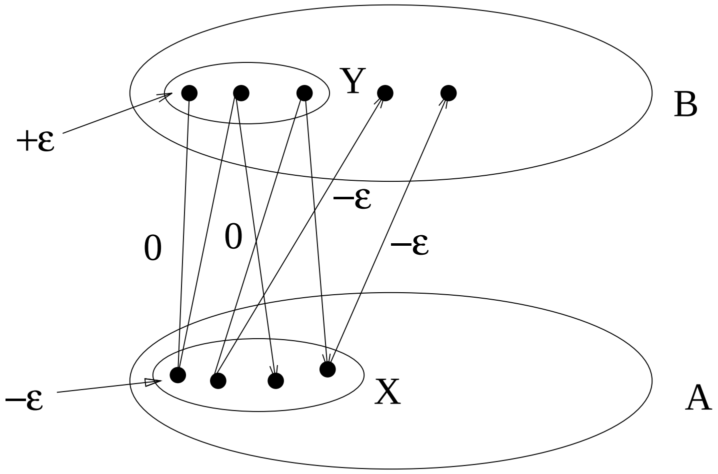

Figure 26: Transforming the weights to improve the decomposition

.

we obtain when running the unweighted bipartite matching algorithm is just $D \cap U$, where D is the set D of the Gallai-Edmonds decomposition of the graph G_0 (the set of outer nodes at the termination of the algorithm), and Y is the set $A \cap W$. Hence no maximum matching of G_0 contains an edge between $U \setminus X$ and Y, and so after adjusting the weights, either (a) the size of the maximum matching in G_0 increases, or (b) it stays the same. Case (a) can occur only $O(n)$ times. In case (b), every element of X remains in D, and so $A \cap W$ increases. So between two consecutive occurences of (a), (b) can occur at most n times.

Thus after at most n^2 iterations we find a 0-weight perfect matching, which is a minimum weight perfect matching in the original graph.

Note that during the whole procedure, we have played around with the weights of edges in a very simple way: we have modified them by adding the same value to each edge adjacent to a give node. So if y_i denotes the total decrease coming from node i, then the final weights of edges are given by

$$w'(ij) = w(ij) - y_i - y_j. \tag{45}$$

We took care of keeping this non-negative, so

$$y_i + y_j \le w(ij)$$

for ever edge ij. Moreover, the final weighting admits a perfect matching with all weights 0, and this is a perfect matching with minimum weight with respect to the original weighting. If $i_1 j_1, \dots, i_n j_n$ is such a perfect matching, then

$$\sum_{k=1}^{n} w(i_k j_k) = \sum_k y_{i_k} + y_{j_k} = \sum_{i \in V} y_i.$$

The $\ge$ sign in this equation follows trivially for any choice of the y_i satisfying (45). Hence as a biproduct of the algorithm, we obtain the following min-max result, due to Egerváry (1931):

Theorem 8.1 *Let $G = (V, E)$ be a complete bipartite graph with equal color classes, and let $w : E \to \mathbb{R}$, a weighting of its edges. Then the minimum weight of a perfect matching in G is equal to the maximum of $\sum_{i \in V} y_i$, where the maximum is extended over all systems of values y_i satisfying $y_i + y_j \le w(ij)$ for every edge ij.*

The weighted matching algorithm can be extended to general (non-bipartite) graphs, but it is discussed more naturally with linear programming at hand, and hence it is postponed now.

8.1.2 The matching polytope of bipartite graphs

We formulate the minimum weight perfect matching problem as a linear programming problem.

Let $G = (V, E)$ be a graph, and assume that each edge e has a weight w_e. With each edge e, associate a variable x_e. Imagine that we have found the minimum weight perfect matching M. We can encode M by setting variable x_e to 1 and 0 depending on whether the edge belongs to the minimum perfect matching or not. We denote the resulting vector (the incidence vector of M) by χ^M. This vector belongs to $\mathbb{R}^E$, the linear space of vectors indexed by the edges of G.

What conditions must this vector χ^M satisfy? The following constraints are easy to guess:

$$\begin{cases} \forall e \quad x_e \geq 0 \\ \forall v \quad \sum_{e \ni v} x_e = 1 \end{cases} \tag{46}$$

Moreover, the weight of the perfect matching is a linear objective function:

$$\text{minimize } \sum_e w_e x_e.$$

The constraints above hold for the incidence vector of any perfect matching; unfortunately, these constraints do not imply that x represents a perfect matching; in fact, a solution may have fractional coordinates. But it is easy to see that this is the only difficulty we can have: if we consider an *integral* solution of (46), then obviously, each variable will be 0 or 1, so the vector will be the incidence vector of some subset M of the edges, and then the second constraint implies that M is a perfect matching.

So we have formulated the minimum weight perfect matching problem as an integer programming problem, which is NP-hard in general. However, this special case is solvable in polynomial time. The case of bipartite graphs is easier, and we start with this.

Let us go ahead and solve the above linear program (46). If we obtain an integer vector as its solutions, we are done; but what happens if the result has fractional entries? The following theorem shows that of we find a *basic solution* (i.e., a vertex of the polytope of solutions), and the graph is bipartite, then this cannot occur.

Theorem 8.2 *If the graph G is bipartite, then the polyhedron $\{x \in \mathbb{R}^E : x_e \geq 0, \sum_{e \ni v} x_e \leq 1\}$ has 0-1 vertices.*

Proof. Let A and B be the color classes of G. We may order the constraints (46) so that the constraints corresponding to nodes in A come first, then come the constraints corresponding to the nodes in B, and finally, the non-negativity constraints. In the matrix N of the system, every column has exactly one 1 in the first $|A|$ rows and one 1 in the next $|B|$ rows (corresponding to the vertices of A and B belonging to the edge). There is an identity matrix on the bottom.

A vertex x of the polyhedron is solution to $Qx = b$, where Q is a non-singular $m \times m$ submatrix of N, and so by Cramer's rule we have

$$x_e = \frac{\det\,(Q_e)}{\det\,Q},$$

where Q_e is formed by replacing the column corresponding to edge e in Q by b.

Lemma 8.1 *Every non-singular square submatrix Q of N has determinant* ± 1.

This implies in particular that every coordinate of a vertex is an integer, and so (since a vertex is a feasible solution) $x_e = 0$ or 1.

Proof. (of the lemma). The assertion is trivially true if Q is 1×1 (since the matrix has only $0/1$ entries).

Given a submatrix Q, we compute its determinant by expanding along appropriate rows and columns. Since each row of identity matrix on the bottom has only one 1, we can expand along those rows if Q contains one, and find that the det Q is equal to the determinant of a smaller submatrix. So suppose that Q contains only rows corresponding to vertices. There must be at least one column with only one 1, otherwise M would be singular (if a column had 0's only, the determinant would be 0; if every column had two 1's, then the sum of the row vectors coming from A would be equal to the sum of the row vectors coming from B and so they would not be independent). So we can expand an use induction as above. ∎

The above results can be stated in a somewhat more general form. A matrix is called *totally unimodular* if every square submatrix has determinant 1, -1 or 0. The lemma shows that the incidence matrix of edges and nodes of a bipartite graph is totally unimodular. Moreover, the argument preceeding it gives the following theorem due to Hoffman and Kruskal.

Theorem 8.3 *Let C be a totally unimodular matrix and c, an integral vector. Then for the problem*

$$\begin{cases} Cx \le c \\ x \ge 0 \\ \ \textit{maximize } w^T x \end{cases}$$

the maximum (if it exists) has integer coordinates.

Theorem 8.2 can be used in combination with the Duality Theorem, and we obtain exactly Egerváry's Theorem 8.1. The values y_i are just the dual variables (corresponding to the primal constraints, which in turn correspond to the nodes). Another remark worth making here concerns the case when the c_{ij} are integers. Since the transpose of a totally unimodular matrix is also totally unimodular, theorem 8.3 implies that the weights y_i can also be restricted to integers.

Exercise 8.1 *Derive König's marriage theorem from this last observation.*

8.2 The non-bipartite matching polytope

We have seen that if G is bipartite then the extremal points of the polyhedron (46) have 0-1 coordinates, but that's not true for a general graph,

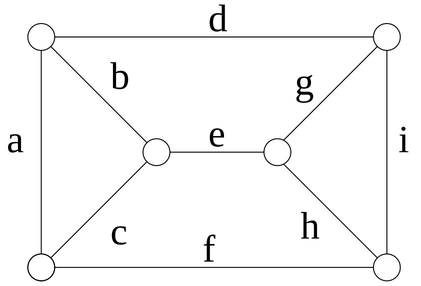

Figure 27: The vertices of the fractional matchiong polytope are not always integral

as the example of figure 27 shows: The point $(1/2, 1/2, 1/2)$ is a vertex of the polytope defined by (46), since it is the unique feasible solution of the system

$$x_d = 0, \qquad x_e = 0, \qquad xx_f = 0,$$
$$x_a + x_b + x_d = 1, \qquad x_b + x_c + x_e = 1, \qquad x_c + x_a + x_f = 1,$$
$$x_g + x_h + x_e = 1, \qquad x_h + x_i + x_f = 1, \qquad x_i + xx_g + xx_d = 1.$$

Exercise 8.2 *Show that for any graph, the vertices of the polyhedron (46) are half-integral.*

The remedy is to add further constraints so as to cut off from the polyhedron the part containing the non-integral vertex or vertices. Let $\text{conv}\{\chi^M\} = PM(G)$ be the *perfect matching polytope* of G. We will try to describe this polytope with inequalities and equations. Clearly, (46) must hold. In addition, we formulate the following constraints: if we partition the vertices into two odd-size sets S and $\overline{S}$, there is at least one matching edge between the sets; thus

$$\sum_{e \in E(S, \overline{S})} x_e \geq 1$$

for any χ^M, and so also for any element of $PM(G)$. The miracle is that these constraints are exactly the ones we need! This is a result due to Edmonds (1965).

Theorem 8.4 *The set of inequalities and equations*

$$\begin{cases} \forall e \;\; x_e \geq 0 & (1) \\ \forall v \;\; \sum_{e \ni v} x_e = 1 & (2) \\ \forall \; \textit{odd } S \textit{ with } 3 \leq |S| \leq n-3, \quad \sum_{e \in E(S,\overline{S})} x_e \geq 1 & (3) \end{cases}$$

defines $PM(G)$.

Proof: We know already that the polyhedron P defined by the 3 sets of constraints contains $PM(G)$. We need to prove that it does not contain any other point. Let x be a vertex of P, and assume that x is not a vertex of $PM(G)$. Then x is not an integral vector. We know that x satisfies at least $|E|$ linearly independent constraints defining P with equality. Let us call these constraints *tight*. Of course, all the constraints (2) are tight.

Claim: At least one tight constraint comes from (3).

Assume not. Consider the graph G^* formed by the edges with positive weight. Because of the constraints in (2), there must be at least one positive x_e adjacent to each vertex. Moreover, if a vertex u has degree 1 in G_0, then the edge uv adjacent to it must have weight 1 (because of (2). Thus there cannot be any other positive edge adjacent to v (because of (2)), and uv is a connected component of G_0 in itself. But then the two constraints of (2) corresponding to u and v, together with the non-negativity constraints, are dependent. So if there are t components of G_0 consisting of a single edge, then the number of linearly independent tight constraints in (1) and (2) is at most $(|E| - |E(G_0)|) + (|V| - t)$, and hence $|E(G_0)! \leq |V| - t$.

Let A be a connected component of G_0 that is not a single edge. Then each node in A is of degree at least 2, and so A has at least $|V(A)|$ edges. Adding this up for all components, we obtain that they contain at least $|V| - t$ edges. Hence every connected component A with more than one edge must have exactly $|V(A)|$ edges, and so it must be regular of degree 2, i.e., a cycle. If A is an odd cycle, then $S = V(A)$ would violate (3); so A is an even cycle. Along the cycle, constraint (2) implies that the weights must be of the form $\alpha, 1-\alpha, \alpha, 1-\alpha, \ldots$. Varying α around its original value, we obtain a segment in $PM(G)$ that contains xx in its interior. But by assumption x is a vertex of the polyhedron, so this is a contradiction. The claim is proved.

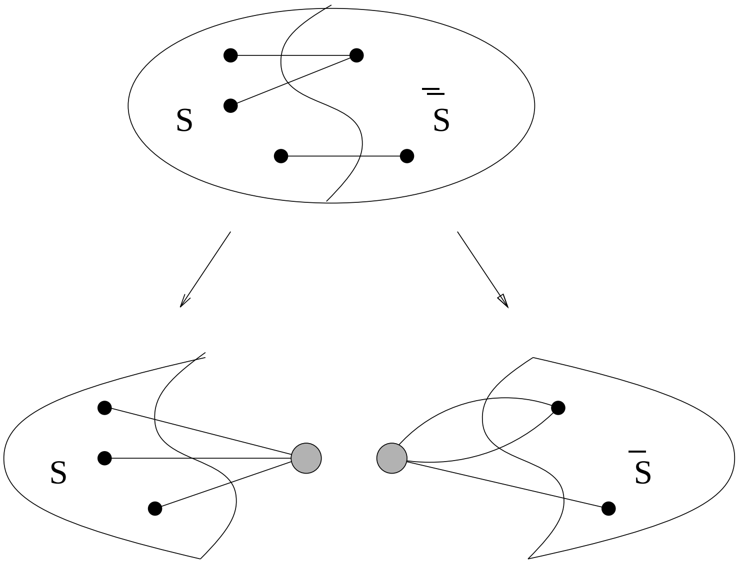

Figure 28: Proof that $P = PM(G)$.

Let S be the set such that $\sum_{e\in E(S,\overline{S})} x_e = 1$. We are going to build two new graphs from G, see figure 28. The first graph is obtained by shrinking $\overline{S}$ to a single point, keeping multiple edges for convenience; the second graph is obtained similarly by shrinking S. Vector x becomes x_1 in G_1, x_2 in G_2. It is easy to see that x_i satisfies constraints (1), (2) and (3) in G_i. Since S and $\overline{S}$ have size at least 3, the graphs G_i are smaller than G, and by induction x_i belongs to $PM(G_i)$. We write

$$\begin{aligned} x_1 &= \sum_{M \text{ pf.match. of } G_1} \lambda_M \chi^M \\ x_2 &= \sum_{N \text{ pf.match. of } G_2} \mu_N \chi^N \end{aligned}$$

where in each sum the coefficients are non-negative and sum to 1. We combine M and N to get a perfect matching $M \cup N$ of G, which is possible when the edge to $\overline{s}$ in G_1 comes from the same edge in G as the edge to s in G_2; thus we get all the matchings of G with exactly one edge accross

$(S,\overline{S})$. Thus

$$x = \sum \frac{\lambda_M \mu_N}{x_e} \chi^{M \cup N}$$

where e is the edge common to M and N. So

$$x = \sum_{e \in E(S,\overline{S})} \sum_{M,N \ni e} \frac{\lambda_M \mu_N}{x_e} \chi^{M \cup N}$$

and it is easy to verify that

$$\sum_{e,M,N} \frac{\lambda_M \mu_N}{x_e} = 1 \ ,$$

thus completing the proof that $x \in PM(G)$. ▮

9 Clutters

9.1 Application to the min-cut problem

Our next example of applications of linear programming techniques will concern the *min-cut* problem.

Let G be a directed graph with a source node s and a sink node t. We assume that there is a capacity $c(e) \geq 0$ on each edge e. Our goal is to partition the vertices into two parts containing s and t respectively, in such a way that

$$\sum_{e: S \to \overline{S}} c(e)$$

is minimum.

Let $\mathcal{C} = \{(s,t) \text{ cuts}\} = \{E(S,\overline{S}) : s \in S, t \in \overline{S}\}$, and for $C \in \mathcal{C}$, let the incidence vector of C be denoted by $\chi^C \in \{0,1\}^E$. The first polytope that comes up to mind is $\text{conv}\{\chi^C, C \in \mathcal{C}\}$. However, this is clearly a bad polytope (with no good separation oracle), since otherwise it could also be used for finding a maximum cut, which is an NP-complete problem.

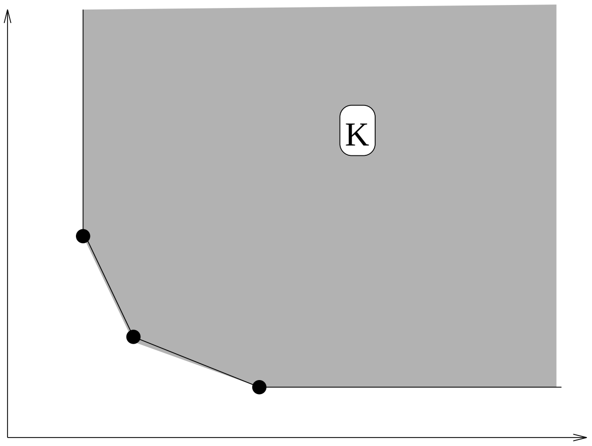

Figure 29: Dominant of the convex hull of a set of vectors.

Thus, we will take instead the *dominant* of the convex hull,

$$K = \text{dom conv}\{\chi^C : C \in \mathcal{C}\} = \{x \in \mathcal{R}^E, \exists x' \in \text{ conv}\{\chi^C\}, x \geq x'\} \ .$$

K is unbounded (see figure 29). How can we find a separation oracle for K? We will use the max-flow min-cut theorem:

Theorem 9.1

$$\max_{f \text{ flow}} v(f) = \min_{C \in \mathcal{C}} c(C) \ ,$$

where the value $v(f)$ of a flow is the sum of the flow on all the edges going out from s.

Given the capacity function c, take a flow $f \leq c$. Then the inequality $v(f) \leq c^T x$ is valid for all $x = \chi^C$, hence for their convex hull and even more so for K. In fact for every flow f the inequality $v(f) \leq f^T x$ is valid for all x in K.

Claim: The inequalities $v(f) \leq f^T x$ define K.

Suppose that the claim is not true. There is an $x_0 \notin K$ such that $\forall f$, $v(f) \leq f^T x_0$. There is a constraint such that $a^T x \geq \alpha$ in K, but $a^T x_0 < \alpha$. The vector a must be ≥ 0, since $a^T x$ must remain greater than α even when, starting from a point χ^C, we increase one of its coordinates by an arbitrarily large amount. For every cut C, we have $a^T\chi^C \geq \alpha$. Hence if we consider a to be the capacity function, we have $c(C) \geq \alpha$: the minimum cut for a has cost at least α. We now apply the max-flow min-cost theorem to a: there exists a flow $f \leq a$ such that $v(f) \geq \alpha$. We have derived the following sequence of inequalities:

$$\alpha > a^T x_0 \geq f^T x_0 \geq v(f) \geq \alpha \ :$$

contradiction. The claim is proved.

Thus K is defined by $\{v(f) \leq f^T x, f \text{ flow}\}$. Many of these constraints are redundant. Since any flow is a positive linear combination of path flows, we can restrict ourselves to (s,t) paths flows P; each edge on the path has flow value equal to 1.

A separation oracle for K then only needs to solve a shortest path problem: given x, we consider its coordinates as edge-lengths, and compute the shortest path from s to t. For a path-flow P, the length of the path is equal to $P^T x$, which must be greater than $v(P) = 1$. Thus if the shortest path has length at least 1, then $x \in K$; otherwise the shortest path P gives a constraint separating x from K.

Another solution consists in realizing that a flow is defined by a polynomial number of inequalities, which, together with the inequalities $f^T x \geq v(f)$, give a linear programming problem, which can be solved in polynomial time with the ellipsoid method. Of course, since the ellipsoid method then calls itself as a subroutine, we get a huge polynomial time; but that's the price we pay for having a really general method which can be applied to many combinatorial optimization problems.

9.2 Clutters: Definition and basic examples

Now we are ready to develop some substantially more general results about min-max theorems and applications of linear programming techniques in combinatorial optimization. First, we need a combinatorial notion and some elementary properties.

Definition 9.1 *A* clutter *is a pair $(E, \mathcal{F})$ such that $\mathcal{F} \subset 2^E$ and $X, Y \in \mathcal{F} \Rightarrow X \not\subset Y$. It is also called an* antichain. *The corresponding* blocking clutter $(E,\mathcal{F}^*)$ *is defined by*

$$\mathcal{F}^* = \{A \subset E,\ \ \forall X \in \mathcal{F}\ \ A \cap X \neq \emptyset, A \ \mathit{minimal}\}.$$

An elementary argument gives the following:

Lemma 9.1 *The blocking clutter of the blocking clutter is the original clutter.*

For example, if E is the set of edges of a graph, and $\mathcal{F}$ the set of $s-t$ dipaths, then we have a clutter. The corresponding blocking clutter consists of minimal $s-t$ cuts, whose blocking clutter, in turn, consists of $s-t$ dipaths.

Clutters and blocking clutters are closely related to minterms and maxterms of monotone boolean formulas: if we consider the clutter $(E,\mathcal{F})$, then the formula

$$f = \bigvee_{A \in \mathcal{F}} (\wedge_{e \in A} x_e)$$

defines a monotone boolean function, and the members of the clutter are the minterms; the members of the corresponding blocking clutter are the maxterms.

We will now see several examples where $E = E(G)$, the set of edges of a graph.

$\mathcal{F} = \{(s,t)$ **dipaths**$\}$ Then an element of the blocker is a minimal set of edges which meets every (s,t) dipath: $\mathcal{F}^* = \{\text{minimal } (s,t) \text{ cuts}\}$.

$\mathcal{F} = \{(s,t)$ **paths**$\}$ in an undirected graph. Then $\mathcal{F}^* = \{\text{minimal } (s,t) \text{ cuts}\}$.

$\mathcal{F} = \{r$-**branchings**$\}$ We recall that a r-branching is a spanning tree whose edges are directed away from the root r. Then $\mathcal{F}^* = \{\text{minimal } r\text{-cuts}\}$, that is, minimal sets $E(S,\overline{S})$, with $r \in S$ and $\overline{S} \neq \emptyset$. Clearly, such a set meets every r-branching. Conversely, given a set Y which meets every r-branching, we consider the vertices which can be reached from r without using edges of Y: if every point can be reached from r, then

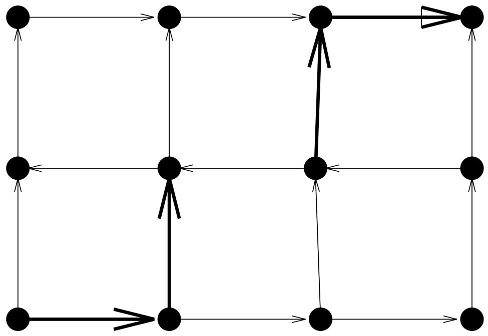

Figure 30: Example of a directed join.

there is a r-branching not using the edges of Y, which is a contradiction; so some points are left unreached from r, and Y contains the cut defined by $S = \{\text{points reachable from } r\}$.

$\mathcal{F} = \{$**spanning trees**$\}$ Then $\mathcal{F}^* = \{\text{minimal cuts}\}$, that is, sets of the form $E(S, \overline{S})$, where S and $\overline{S}$ are both non-empty.

$\mathcal{F} = \{$**circuits**$\}$ in an undirected graph. Then $\mathcal{F}^* = \{\text{complement of a spanning tree}\}$. We note that this actually holds for any matroid.

$\mathcal{F} = \{$**directed circuits**$\}$ Then an element of the blocker must be the complement of a acyclic subgraph. $\mathcal{F}^* = \{\text{feedback sets}\}$.

$\mathcal{F} = \{$**directed joins**$\}$ where in a graph which is not strongly connected, a *directed join* is defined to be any minimal set of edges whose contraction makes the graph connected (see example figure 30). From this viewpoint, contracting an edge can also be seen as adding a parallel edge in the opposite direction. Then we claim that $\mathcal{F}^* = \{\text{directed cuts}\}$, where a *directed cut* is a partition of the vertices into two sets S and $\overline{S}$ such that all the edges between S and $\overline{S}$ go from S to $\overline{S}$. Clearly, given a directed cut, a join must contain one of its edges. The converse is not quite obvious. Consider the clutter $\mathcal{G} = \{\text{directed cuts}\}$. We will show that $\mathcal{G}^* = \mathcal{F}$, hence $\mathcal{G} = \mathcal{F}^*$. Take a set of edges which meets every directed cut. Contract those edges in the set. The contracted graph has no directed cut, hence is strongly connected; hence the set must contain a directed join, QED.

9.3 The Chinese Postman problem

The *Chinese Postman* problem was raised by the Chinese mathematician Meigu Guan in 1962, and solved by Edmonds and Johnson in 1973. Given an connected undirected graph and a non-negative edge cost function $c(e)$, the postman wants to start from the post-office, visit all the edges and come back to the post-office walking minimum total length. It is well-known that if the graph is eulerian (all vertices have even degree), then there is a circuit which visits each edge exactly once, and the optimal length is the total length of the edges.

For a general graph, we claim that a minimal tour will never visit an edge more than twice. In fact, the minimal tour can be seen as adding some parallel edges to existing edges so as to make the graph eulerian. If there are 3 parallel edges between two points, we can delete two of them, and the degrees of the vertices will keep the same parity, QED.

Thus the length of a minimal tour is $\sum_{e \in E} c(e) + \sum_{e \in \mathcal{J}} c(e)$, where $\mathcal{J}$ is the set of edges through which the tour goes twice.

We can generalize the problem in the following way: given a mapping $t : V(G) \to \{0, 1\}$, find a subset $\mathcal{J}$ of the edges, such that for every vertex $\deg_{\mathcal{J}}(v) \equiv t(v) \pmod 2$ and which minimizes the total length $c(\mathcal{J})$.

Definition 9.2 *A set $\mathcal{J}$ which satisfies those parity conditions and is minimal for inclusion is called a t-*join.

Assume that G is connected. When does there exist a t-join at all?

Claim: A connected graph has a t-join if and only if there is an even number of nodes with parity 1 (in other words, $\sum_v t(v)$ is even).

In fact, a t-join can then be constructed in the following way: we pair the nodes who map to 1 through t. For each pair, we take a path connecting them. If we add all these paths modulo 2, it is easy to see that we obtain a t-join. Moreover, since deleting a cycle does not change the parities at any node, a minimal t-join must be a forest.

In the special case where t is the constant 1 function, every perfect matching is a 1-join, and every 1-join with $n/2$ edges is a perfect matching. Any 1-join has at least $n/2$ edges, but there are 1-joins with more that $n/2$ edges.

If we have an algorithm for finding a minimum 1-join, we can use it to find a minimum weight perfect matching as follows. Adding some large constant to the cost function before running the algorithm will heavily penalize the 1-joins with many edges, so that the algorithm will output a perfect matching (if it exists), and since adding a constant to all the edges does not change the relative cost of the perfect matchings, the algorithm actually outputs a minimum weight perfect matching.

Conversely, we will now show that a minimum weight t-join problem can be reduced to a minimum weight perfect matching problem. Let T denote the set of vertices v such that $t(v) = 1$; pair the vertices of T in some arbitrary way, (x_i, y_i). Let $d(x_i, y_i)$ be the length of the shortest path from x_i to y_i in G. Since adding these paths gives a t-join, we get that the minimal cost of a t-join is at most $\sum_i d(x_i, y_i)$. Choosing the pairing which minimizes this sum amounts to solving an instance of a minimum weight perfect matching problem on the complete graph on T, with cost function $c(x, y) = d(x, y)$ in G.

To prove that this produces the optimum t-join, consider an arbitrary t-join; it is a forest. Delete the edges of a path in the forest connecting two endnodes, and repeat (see figure 31). Thus the t-join can be decomposed as a union of edge-disjoint paths between vertices of T, and its cost, equal to the sum of the lengths of these paths, is at least the sum of the lengths of the shortest paths:

$$c(t\text{-join}) \geq \sum_i d(\overline{x_i}, \overline{y_i}),$$

which in turn is an upper bound on the minimum perfect matching cost, hence the proof.

Hence there is a polynomial-time algorithm for solving the Chinese postman problem. Finding a polynomial time algorithm for the general version of

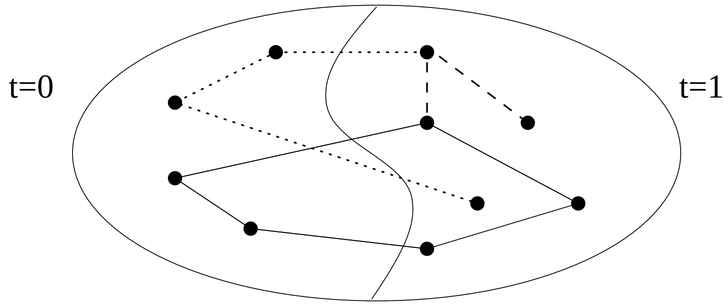

Figure 31: Defining a matching from a t-join

the problem (for general t) implies in turn an alternate algorithm for the minimum weight perfect matching problem.

The blocking clutter of $\mathcal{F}$ is the set of minimal cuts such that $t(S)$ is odd, called *t-cuts* by definition. Thus $\mathcal{F}^* = \{t\text{-cuts}\}$.

9.4 The Max-Flow-Min-Cut property for clutters

9.4.1 Definition

If we look at all our examples of clutters, we see that many combinatorial optimization problems can be formulated in the following way: given a clutter $(E, \mathcal{F})$ and a non-negative cost function $c : E \to \mathcal{R}_+$, minimize $\{c(X), X \in \mathcal{F}\}$.

Unfortunately, we do not have any hope of solving this problem in its full generality: that would imply a polynomial solution for the min feedback set problem, which is NP-complete. But we can try to apply linear programming techniques.

We showed that this is equivalent to calculating

$$\min\{c \cdot x : x \in \mathcal{P}(\mathcal{F})\}$$

where c is now thought of as a vector in $\mathcal{R}^E$ and

$$\mathcal{P}(\mathcal{F}) = \text{dom conv}\{\chi^A : A \in \mathcal{F}\}.$$

We would like to put this minimization problem into a linear programming form so that we can use our knowledge of linear programming techniques to solve it. This means that we have to be able to express $\mathcal{P}(\mathcal{F})$ by linear inequalities. What are these inequalities? Natural ones that any $x \in \mathcal{P}(\mathcal{F})$ has to obey are

$$\begin{gathered} x > 0 \\ \forall B \in \mathcal{F}^*, \quad x \cdot \chi^B \geq 1. \end{gathered}$$

These equations follow directly from the definition of $\mathcal{P}(\mathcal{F})$. If $x \in \mathcal{P}(\mathcal{F})$ then $x \geq y$ for some $y \in \operatorname{conv}\{\chi^A : A \in \mathcal{F}\}$. Therefore there exist weights $\lambda_A \geq 0$, $\sum_{A \in \mathcal{F}} \lambda_A = 1$ such that $x \geq \sum \lambda_A \chi^A$. For every $B \in \mathcal{F}^*$ and every $A \in \mathcal{F}$ we have $\|A \cap B\| \geq 1$ because B blocks A. Therefore $\chi^A \cdot \chi^B \geq 1$. But then

$$x \cdot \chi^B \geq \sum_{A \in \mathcal{F}} \lambda_A (\chi^A \cdot \chi^B) \geq \sum_{A \in \mathcal{F}} \lambda_A = 1.$$

In some cases the blocking clutter inequalities are not only necessary but sufficient, a fact which motivates the following definition:

Definition 9.3 *A clutter* $(E, \mathcal{F})$ *has the* Max-Flow-Min-Cut (MFMC) *property if*

$$\mathcal{P}(\mathcal{F}) = \{x : x \geq 0, \quad \forall B \in \mathcal{F}^*, \ x \cdot \chi^B \geq 1\}$$

Before discussing why this property helps us solve the clutter minimization problem we are going to look at an easy lower bound on a minimal cost clutter. Let $\{\lambda_B\}$ be any good "packing" of the blocking clutters. We mean by this that λ_B of $\mathcal{F}^*$ that has the following properties:

$$\begin{gathered} \forall B \in \mathcal{F}^*, \ \lambda_B \geq 0 \\ \forall e \in E, \ \sum_{B \ni e} \lambda_B \leq c(e). \end{gathered}$$

For any $A \in \mathcal{F}$ we have

$$\sum_{e \in A} \sum_{B \ni e} \lambda_B \leq \sum_{e \in A} c(e) = c(A)$$

On the other hand the fact that $\|A \cap B\| \geq 1$ gives

$$\sum_{B\in\mathcal{F}^*} \lambda_B \leq \sum_{B\in\mathcal{F}^*} \lambda_B \sum_{\substack{e\in A\\ e\in B}} 1 \leq \sum_{e\in A} \sum_{B\ni e} \lambda_B .$$

We have just shown

$$\forall A \in \mathcal{F},\ \sum_{B\in\mathcal{F}^*} \lambda_B \leq c(A).$$

Since λ and A were arbitrary we have actually just proven the following lower bound on the minimal weight clutter.

$$\max_{\lambda}\{ \sum_{B\in\mathcal{F}^*} \lambda_B\} \leq \min\{c(A) : A \in \mathcal{F}\}.$$

Suprisingly it is the case that those clutters where equality holds in the above formula are exactly those clutters which possess the *MFMC* property, hence its name.

Theorem 9.2 *A clutter* $(E, \mathcal{F})$ *possesses the* Max-Flow-Min-Cut *property if and only if* $\forall c : E \to \mathcal{R}_+$

$$\max_{\lambda} \sum_{B\in\mathcal{F}^*} \lambda_B = \min\{c(A) : A \in \mathcal{F}\}.$$

The proof is not difficult. The *only if* direction is a mechanical application of the duality theorem of linear programming (see *Combinatorial Optimization* by Papadimitriou and Steiglitz, pages 67-69 for details). If the clutter has the *MFMC* property then the minimization problem is just the linear program

$$\begin{aligned} \min c \cdot x \\ x \geq 0 \\ \forall B \in \mathcal{F}^*,\ \ x \cdot \chi^B \geq 1. \end{aligned}$$

Its dual is

$$\begin{aligned} \max \sum \lambda_B \\ \forall B \in \mathcal{F}^*,\ \lambda_B \geq 0 \\ \forall e \in E,\ \sum_{B\ni e} \lambda_B \leq c(e). \end{aligned}$$

and the duality theorem tells us that the optimal solutions of an LP and its dual are equal.

To prove the *if* direction we have to show that, if $\forall c : E \to \mathcal{R}_+$ that

$$\max_{\lambda} \sum_{B \in \mathcal{F}^*} \lambda_B = \min\{c(A) : A \in \mathcal{P}(\mathcal{F})\}$$

then $\mathcal{P}(\mathcal{F})$ has the *MFMC* property. Let

$$\mathcal{Q}(\mathcal{F}) = \{x : x \geq 0, \quad \forall B \in \mathcal{F}^*, \ x \cdot \chi^B \geq 1\}.$$

Applying the duality theorem of linear programming to the dual LPs given above and explicitely writing out the *MFMC* property we see that this is equivalent to proving that if $\forall c : E \to \mathcal{R}_+$

$$\min\{c \cdot x : x \in \mathcal{P}(\mathcal{F})\} = \min\{c \cdot x : x \in \mathcal{Q}(\mathcal{F})\}$$

then

$$\mathcal{P}(\mathcal{F}) = \mathcal{Q}(\mathcal{F}).$$

Suppose that $\mathcal{P}(\mathcal{F}) \neq \mathcal{Q}(\mathcal{F})$. We saw before that $\mathcal{P}(\mathcal{F}) \subseteq \mathcal{Q}(\mathcal{F})$ so $\exists x \in \mathcal{P}(\mathcal{F}) - \mathcal{Q}(\mathcal{F})$. We will use this x to find $c \geq 0$ such that

$$\min\{c \cdot x : x \in \mathcal{P}(\mathcal{F})\} < \min\{c \cdot x : x \in \mathcal{Q}(\mathcal{F})\}$$

Since $\mathcal{P}(\mathcal{F})$ is convex there is a hyperplane through x that separates it from $\mathcal{P}(\mathcal{F})$. Analytically this implies the existence of a vector c such that

$$\forall y \in \mathcal{P}(\mathcal{F}) \ \ c \cdot y > c \cdot x.$$

Suppose c has a negative component c_i. Choose a point $y \in \mathcal{P}(\mathcal{F})$. Fixing all other components of y we can arbitrarily increase its $i'th$ component while still keeping $y \in \mathcal{P}(\mathcal{F})$ (by the definition of *dom*). It follows that $\min\{c \cdot y : y \in \mathcal{P}(\mathcal{F})\} = -\infty$ contradicting our choice of c. Therefore $c \geq 0$ and we are done.

9.4.2 Examples

The cannonical example of a clutter where the theorem is applicable is *min-cuts*. Given a digraph $G = (V, E)$ with a source s, a sink t, and positive weights, c, on the edges, we define

$$\mathcal{F} = \{E(S, \bar{S}) : S \ni s, \ \bar{S} \ni t\}$$

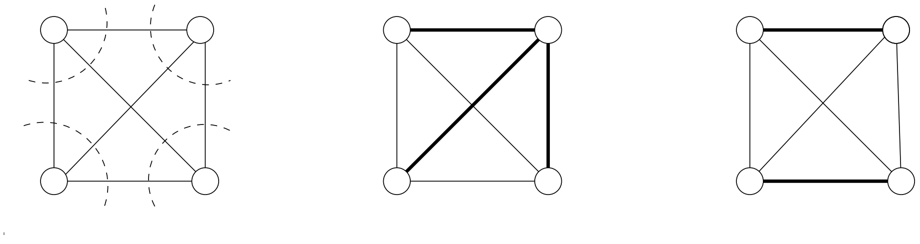

Figure 32: t-cuts and joins on K_4

and it is not difficult to see that

$$\mathcal{F}^* = \{B : B \text{ is a simple directed } s\text{-}t \text{ path}\}.$$

(As always when discussing clutters there is an implicit assumption that the definition of $\mathcal{F}$ requires minimality by inclusion). The max-flow-min-cut theory of network flows tells us that

$$\max_f \sum_{(s,u)\in E} f(s,u) = \min\{c(A) : A \in \mathcal{F}\}$$

where the maximum is taken over all legal flows. Any flow can be naturally decomposed into the disjoint sum of flows over s-t paths (the path flows are disjoint, the paths themselves don't have to be). Take such a decomposition of a max flow. Set λ_B to be the flow along that path in the decomposition. Notice that the flow out of the sink is $\sum_{B\in\mathcal{F}^*} \lambda_B$ and that, by the definition of flows, $\forall e \in E \ \sum_{B\ni e} \lambda_B \leq c(e)$. Conversely, given a legal labeling λ_B we can construct a legal flow $f(e) = \sum_{B\ni e} \lambda_B$. We've just shown that flows correspond to good labelings of s-t paths so

$$\max_\lambda \sum_{B\in\mathcal{F}^*} \lambda_B = \min\{c(A) : A \in \mathcal{F}\}$$

and by the theorem the *min-cut* clutter has the MFMC property.

An important fact to note is that even though the primal LP is integral (since it is defined by incidence vectors) optimality of the dual (λ_B) doesn't necessarily occur at integral points. As an example we consider the 1-join on the complete graph K_4 (figure 9.4.2). In this graph every minimum weight 1-join is a minimum weight perfect matching and therefore has weight 2. The t-cuts (blocking clutters) are the four stars. There is no integral labelling λ_B of the t-cuts such that $\sum \lambda_B = 2$. In fact the only optimal labelling is the one that assigns each t-cut the value $1/2$.

A more complicated example, one in which we can find a maximal labeling, is given by 1-joins on bipartite graphs that don't have perfect matchings. Formally suppose $G = (V, E)$ has a bipartition U, V (ie. $E \subseteq U \times V$) where $\|A\| = \|B\| = n/2$, *and* G has no perfect matching. Given the weight function $c \equiv 1$ the problem is to find a minimum weight 1-join of G. Since G has no perfect matching this means that the minimum weight 1-join has weight $> n/2$. Assuming for the moment that we know that 1-joins have the *MFMC* property (a fact we will not prove; we shall give the proof of an analogous result on dijoins) this tells us that the maximum labeling of the blocking clutters has $\sum \lambda_B > n/2$.

In fact we can explicitely construct a labeling that has this property. First note that $\forall v \in G$ the set $\{e : e \ni v\}$ is a blocking set (it might be necessary to take some minimal subset of it). Therefore a nonnegative labeling λ_v of the vertices such that $\forall (u, v) \in E,\ \lambda_u + \lambda_v \leq 1$ is a valid labeling of the blocking clutters (any blocking clutter that is not contained in some $\{e : e \in v\}$ will be labeled 0). Since there is no perfect matching Hall's theorem tells us that there is some $X \subset U$ such that $\|\Gamma(X)\| < \|X\|$ where $\Gamma(X)$ is the neighbor set of X. Label the vertices as follows:

$$\begin{aligned} \lambda_u &= 1, & u &\in X \\ \lambda_u &= 0, & u &\in U - X \\ \lambda_v &= 0, & v &\in \Gamma(X) \\ \lambda_v &= 1, & v &\in V - \Gamma(X). \end{aligned}$$

This is a good labeling which is integral and whose sum is $> n/2$. A more complicated argument due to Seymour shows that there is some *maximal* integral labelling.

9.4.3 Duality properties: Lehman's theorem and applications

We will now prove a theorem due to Lehman, that the *MFMC* property is invariant under duality.

Theorem 9.3

$(E, \mathcal{F})$ *has the* MFMC *property* $\Rightarrow$ $(E, \mathcal{F}^*)$ *has the* MFMC *property.*

We are given that

$$\mathcal{P}(\mathcal{F}) = \{x : x \geq 0,\ \ \forall B \in \mathcal{F}^*,\ x \cdot \chi^B \geq 1\}$$

and we want to prove that

$$\mathcal{P}(\mathcal{F}^*) = \{x : x \geq 0,\ \ \forall A \in \mathcal{F},\ x \cdot \chi^A \geq 1\}.$$

The discussion in the previous section tells us that $\mathcal{P}(\mathcal{F}^*) \subseteq \{\cdots\}$ so we only have to prove that if $y \geq 0,\ \forall A \in \mathcal{F},\ y \cdot \chi^A \geq 1$ then $y \in \mathcal{P}(\mathcal{F}^*)$. Suppose by contradiction that there is some $y \notin \mathcal{P}(\mathcal{F}^*)$ that satisfies all of these inequalities. The convexity of $\mathcal{P}(\mathcal{F}^*)$ tells us that there is a hyperplane which separates y from $\mathcal{P}(\mathcal{F}^*)$ so

$$\exists c \in \mathcal{R}^E,\ c \cdot y < \min\{c \cdot x : x \in \mathcal{P}(\mathcal{F}^*)\}.$$

Furthermore the same argument as was used in the second part of the proof of the theorem shows that $c \geq 0$. Without loss of generality we can assume that

$$\min\{c \cdot x : x \in \mathcal{P}(\mathcal{F}^*)\} = 1$$

(if not we scale c, dividing by the minimum value of $c \cdot x$ in $\mathcal{P}(\mathcal{F}^*)$). Thus

$$\forall x \in \mathcal{P}(\mathcal{F}^*),\ c \cdot x \geq 1$$

and more specifically

$$\forall B \in \mathcal{F}^*,\ c \cdot \chi^B \geq 1.$$

We are given that $\mathcal{F}$ has the *MFMC* property and so the last statement says that $c \in \mathcal{P}(\mathcal{F})$. This means that c dominates some convex combination of the χ^A. Since y was chosen so that $\forall A \in \mathcal{F},\ \chi^A \cdot y \geq 1$ this gives $c \cdot y \geq 1$. But $c \cdot y < 1$ and so we have reached a contradiction and are done.

We will use this theorem to rederive the original Max-Flow-Min-Cut from basic principles. We saw before that if

$$\mathcal{F} = \{B : B \text{ is a simple directed } s\text{-}t \text{ path}\}$$

then

$$\mathcal{F}^* = \{\ s\text{-}t \text{ cuts}\}.$$

For any weight function c define the distance function on E:

$$d(u) = \text{ weight of the minimal weight path from } s \text{ to } u.$$

Given any α such that $0 \leq \alpha < d(t)$ it is easy to see that the set

$$C_\alpha = \{(u,v) : d(u) \leq \alpha,\ d(v) > \alpha\}$$

blocks $\mathcal{F}$. Therefore we can find $B_\alpha \in \mathcal{F}^*$ such that $B_\alpha \subseteq C_\alpha$. Next we order the points in E so that

$$d(s = v_0) \leq d(v_1) \leq d(v_2) \leq \cdots \leq d(t)$$

(ignoring all points which are further from s than t is). This gives us some ordered sequence of distances

$$0 = d_0\ < d_1 < d2 < \cdots < d_k = d(t).$$

Label $\mathcal{F}^*$ as follows. If for some $0 \leq i < k$ we have $B = B_{d_i}$ then set

$$\lambda_B = d_{i+1} - d_i,$$

otherwise set $\lambda_B = 0$. The definition of $d(u)$ guarantees that this is a good labeling. Furthermore this labeling fufills the right hand side condition of the theorem since

$$\sum \lambda_B = d(t) = \min\{c \cdot x : x \in \mathcal{P}(\mathcal{F})\}.$$

Therefore $(E, \mathcal{F})$ has the *MFMC* property and by Lehman's theorem so does $(E, \mathcal{F}^*)$. Another application of the theorem shows that $\forall c : E \to \mathcal{R}_+$

$$\max_\lambda \sum_{A \in \mathcal{F}} \lambda_A = \min\{c(B) : B \in \mathcal{F}^*\}$$

which, as discussed following the proof of theorem 18.1, is actually the classic Max-Flow-Min-Cut theorem of network flows. (Note: Try not to be confused by the fact that $\mathcal{F}$ and $\mathcal{F}^*$ were switched in the two discussions. In the first $\mathcal{F}$ was the cuts while in the second it was the paths.)

9.5 Binary clutters

The final topic that we will address is that of binary clutters.

Definition 9.4 *A clutter $(E, \mathcal{F})$ is a* binary clutter *if* $\forall X, Y, Z \in \mathcal{F}$

$$\exists W \text{ s.t. } X \oplus Y \oplus Z \supseteq W \in \mathcal{F}$$

where $\oplus$ is mod two union ($e \in X \oplus Y$ iff $e \in X - Y$ or $e \in Y - X$).

As an example let $\mathcal{F}$ be the clutter consisting of odd length circuits in a graph. Then $X \oplus Y \oplus Z$ is going to be the union of cycles, one of which is odd, so $\mathcal{F}$ is a binary clutter.

t-cuts also form a binary clutter. Recall that a t-cut can be expressed as $E(S, \bar{S})$ where the parity $p(S) = \sum_{v \in S} t(v) \pmod 2$ is 1. In order to prove that t-cuts form a binary clutter we will first show that the mod 2 union of two t-cuts is a cut $E(V, \bar{V})$ where $p(V) = 0$. Then we will show that the mod 2 union of such a cut with a t-cut is again a t-cut.

Given any two cuts (not necessarily t-cuts) $E(S, \bar{S})$ and $E(T, \bar{T})$ it is not difficult to see that their mod 2 union is also a cut;

$$E(S, \bar{S}) \oplus E(T, \bar{T}) = E(S \oplus T, \overline{S \oplus T})$$

(where $S \oplus T = (S \cap T) \cup (\bar{S} \cap \bar{T})$). This happens because any edges from (for example) $S \cap T$ to $\bar{S} \cap \bar{T}$ will appear twice and thus be erased. Suppose now that we are given two t-cuts $E(S, \bar{S})$ and $E(T, \bar{T})$. Without loss of generality we can assume that $p(S \cap T) = 1$ (otherwise switch T and $\bar{T}$). Then $p(S \cap \bar{T}) = p(\bar{S} \cap T) = 0$ and $p(\bar{S} \cap \bar{T}) = 1$. Setting $V = S \oplus T$ this shows that

$$E(S, \bar{S}) \oplus E(T, \bar{T}) = E(V, \bar{V})$$

where $p(V) = 0$. Now suppose that $E(U, \bar{U})$ is another t-cut. Again without loss of generality we can assume that $p(U \cap V) = 1$ (otherwise switch U and $\bar{U}$). Then $p(\bar{U} \cap V) = p(\bar{U} \cap \bar{V}) = 0$ and $p(U \cap \bar{V}) = 1$. Therefore $p(U \oplus V) = 1$ and the cut

$$E(U, \bar{U}) \oplus E(V, \bar{V})$$

is a t-cut.

It is possible to prove that the blocker of a binary clutter is also a binary clutter. It is not always true that a binary clutter has the *MFMC* property. In fact Seymour has proven that those binary clutters that do have the *MFMC* property are compositions (in a complicated but well defined way) of t-joins and t-cuts. Finally, there is a (vague) conjecture that every *clutter* that has the *MFMC* property is somehow constructable from t-cuts, directed cuts, rooted cuts, perhaps other kinds of cuts and their blockers.

Seymour gave a characterization of those binary clutters that do have this property. To state his result, we extend the definition of deletion and contraction from matroids to clutters. The *deletion* of an element $v \in E$ results in the clutter $(E \setminus v, \mathcal{F} \setminus v)$, where $\mathcal{F} \setminus v)$ consists of the minimal sets X such that $v \notin X$ and $X \in \mathcal{F}$. The *contraction* of v results in the clutter $(E \setminus v, \mathcal{F}/v)$, where $\mathcal{F}/v)$ cvonsists of the minimal sets X such that $v \notin X$ and $X \cup v \in \mathcal{F}$. A *minor* of a clutter is any clutter obtained by deleting and contracting elements. We denote by Q_6 the clutter consisting of the odd circuits of K_4.

Theorem 9.4 . *A binary clutter has the Max-Flow-Min-Cut property iff it has no* Q_6 *minor.*

The proof of this theorem is difficult. Using this result and some deep matroid theory, Truemper gave another characterization of binary clutters with the MFMC property: they are compositions (in a complicated but well-defined way) into t-cut clutters and t-join clutters. Finally, there is a (vague) conjecture that every clutter with the MFMC property can be "glued together" from t-cuts, directed cuts, rooted cuts, perhaps other kinds of cuts, and their blockers.

10 The Lucchesi-Younger theorem

10.1 The Dual Integrality theorem

We recall that if we consider a linear program and its dual,

$$\begin{cases} Ax \leq b \\ x \geq 0 \\ \text{maximize } c^T x \end{cases} \qquad \begin{cases} yA \geq c \\ y \geq 0 \\ \text{minimize } yb \end{cases}$$

then if both have a solution we have equality $\max c^T x = \min yb$ by the Duality Theorem of linear programming. However this is no longer true if we restrict ourselves to integer solutions. We have:

$$\max_{x \text{ integer}} c^T x \leq \max c^T x = \min yb \leq \min_{y \text{ integer}} yb \ .$$

One or both of the inequalities may be strict, as can be seen by taking the example from above with the K_4 graph and the clutter of 1-joins. Since it possesses the MFMC property, finding a minimal 1-join is the same as minimizing $c^T x$ under the constraints $x \geq 0$ and $x(C) \geq 1$ for every 1-cut C. The minimum 2 is achieved for integral x, but the maximum of the dual problem (finding a maximum labeling) is not reached for integral y.

Yet, whether the two inequalities are strict or not are not independent, as the following theorem shows.

Theorem 10.1 *(Hoffman, Edmonds-Giles) If the dual program has an integral optimum solution for every integral c, then so does the primal (assuming A and b integral).*

Note that the theorem is not symmetric under duality; thus in our example we do not have equality for the dual problem, and the theorem does not apply.

The proof uses separation properties as usual. Let P be the polyhedron $\{x \geq 0, Ax \leq b\}$. We want to prove that every vertex of P is integral (which is equivalent to saying that P has integral solutions for every c). Through a vertex v of P, there exists a hyperplane which intersects P at v only. Let c_0 be the vector orthogonal to that hyperplane. We can change its direction slightly so that its coordinates become rational; v has rational coordinates since they are solution of a set of equations with integer coeeficients. Thus c_0 is rational, and scaling by a suitably large constant makes c_0 an integer. There exists an integer N large enough so that for all i the vector $c_i = Nc_0 + e_i$ still defines a hyperplane which intersects P at v only. Since ${c_i}^T v = \max_P {c_i}^T x = \min yb$, which is integral by our hypothesis on the dual program $(0 \leq i \leq n)$, we have $v_i = e_i^T v = c_i^T v - c_0^T v$ also integral. So every vertex of P has integral coordinates.

We can extend the result to slightly more general linear programs without non-negativity constraints. The difficulty is then that P may not have any vertices at all (if it has a vertex then every linear objective function is maximized by a vertex, and the previous proof applies). If P has no vertices then we can argue that it contains at least one line with rational coordinates, and we can apply the proof to the projection of P along that direction. (We shall not, however, meet such linear programs in our combinatorial applications.)

Our first application concerns matchings of bipartite graphs. Let P be the convex hull of the characteristic vectors of the matchings (a polyhedron in $\mathcal{R}^E$). We proved that P was defined by the inequalities $x_e \geq 0$ and $\forall v$, $\sum_{e \ni v} x_e \leq 1$. Can we apply the dual integrality theorem? The dual program, for any integral edge cost function c, is of the form:

$$\begin{cases} \forall v \;\; y_v \geq 0 \\ \forall e = (uv), \;\; y_u + y_v \geq c(e) \\ \text{minimize } \sum_v y_v \end{cases}$$

The optimum solution is rational (it is a vertex of a polyhedron defined by rational constraints). Take the optimum solution which has as many integer coordinates as possible. Each part of the graph is partitioned into two subsets depending on whether y_v is integer or not at vertex v; let ϵ be a small positive value. Assume that A has at least as many vertices with non-integer values y_v as B; we claim that we get a new solution by substracting ϵ to the non-integer y_v's of A and adding ϵ to the non-integer y_v's of B (see figure 33). For a suitable choice of ϵ the constraints $y_v \geq 0$ are still met. The edge constraints are unchanged if e connects two points with integer y_v, or two points with non-integer y_v; if it connects a point with integer y_v with a point with non-integer y_v, the constraint

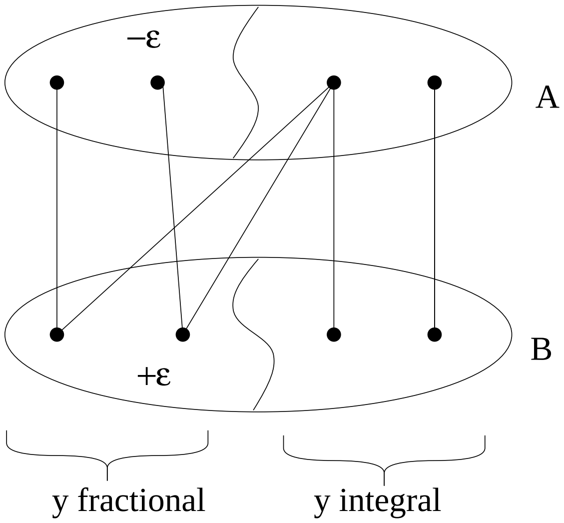

Figure 33: Applying the TDI theorem to bipartite matchings

was not met with equality and so for a suitable choice of ϵ the constraint is still met. Thus we do have a new solution which is at least as good as the original one, and increasing ϵ continuously we eventually satisfy one more constraint with equality, and the new solution has more integral coordinates. We have reached a contradiction.

Thus the hypotheses of the total dual integrality theorem are satisfied and P has an integral optimum solution for every integral c.

10.2 The Lucchesi-Younger theorem

We will now study another application of the theorem, to the problem of finding a minimum directed join in a directed graph. Let $G = (V, E)$ be a directed graph, $\mathcal{F}$ be the set of directed joins (minimal sets of edges whose contraction makes G strongly connected). We know that $\mathcal{F}^*$ is the set of minimal directed cuts. We have the following useful theorem:

Theorem 10.2 *(Lucchesi-Younger) The minimal size of a directed join is equal to the maximum number of edge-disjoint directed cuts.*

This theorem has some implications on the feedback number of planar graphs: finding the feedback number of a general graph is NP-hard, but for a directed planar graph G we can draw its planar directed dual (where a dual edge is oriented so as to cross the corresponding primal edge from left to right). We are going to prove that the feedback number of G is the minimum size of a directed join of G^*.

Deleting an edge of G is equivalent to contracting the dual edge of G^*. If there is a cycle in G, then all the dual edges of the cycle edges cross the cycle going inside (or all going outside), so they form a directed cut of G^*, which is therefore not strongly connected. Conversely, if G^* is not strongly connected, then contracting each connected component makes a directed acyclic graph, which must have at least one vertex of indegree 0; the corresponding strong connected component in G^* has all edges connecting it to the rest of the graph oriented outward, which means that the dual edges in G form a directed cycle. Thus G has a cycle iff G^* is not strongly connected, and deleting edges of G until it becomes an acyclic graph is equivalent to contracting edges of G^* until it becomes strongly connected: the feedback number of G is the minimum size of a dijoin of G^*.

This property used jointly with the Lucchesi-Youneger theorem gives the following result:

Corollary 10.1 *In a planar graph, the feedback number is equal to the maximum number of edge-disjoint directed cycles.*

Of course the $\geq$ direction of the theorem is always true. For general graphs the result no longer holds, as can be seen from the example figure 34. With the orientations of the figure, there are no two edge-oriented directed cycles, but there is no way to delete just one edge and destroy all the cycles. It is conjectured that for a general graph if there are no two edge-disjoint directed cycles, then the feedback number is at most 4.

Coming back to investigating the power of the Lucchesi-Younger theorem, we are going to show that it implies that the clutter of directed joins has the MFMC property.

In the case where the edge costs are all equal to 1, the MFMC theorem states that

$$\min\{|A|, A \in \mathcal{F}\} = \max\{\sum_{\mathcal{F}^*} \lambda_B : \lambda_B \geq 0, \sum_{B \ni e} \lambda_B \leq 1\}$$

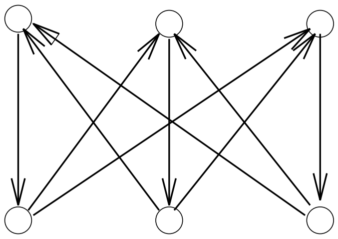

Figure 34: $K_{3,3}$ has a high feedback number

and choosing λ_B equal to 1 for the set of edge-disjoint dicuts, and 0 otherwise, gives the MFMC equality from the theorem.

In the case where the costs are all integers, we can replace each edge $e = uv$ by a path formed by $c(e)$ edges $uv_1, v_1v_2, \ldots, v_{c(e)-1}v$; contracting e is replaced by contracting the $c(e)$ corresponding edges; cuts through e, in number at most $c(e)$, correspond to edge-disjoint cuts through the path; thus the theorem still implies the MFMC property.

Homogeneity in c means that the theorem must also imply MFMC when c is rational, and finally by continuity the theorem must imply MFMC for all real cost functions c.

Conversely, if we know that we have the MFMC property and that the λ_B's are integral, then we can deduce the theorem. Thus the Lucchesi-Younger theorem is equivalent to the MFMC property with integral λ.

10.3 Proof of the Lucchesi-Younger theorem

In order to prove the theorem, we will show that the "dual" program

$$\begin{cases} \sum_{B \ni e} y_B \le c(e) \\ y_B \ge 0 \\ \max \sum y_B \end{cases}$$

is optimized by an integral vector, for all non-negative cost function c (notice that if c has a negative coordinate, then the minimum is $-\infty$ and the program is unfeasible); then we can apply the Hoffman-Edmonds-Giles theorem seen last class, to obtain that the polyhedron belonging to the "primal" program

$$\begin{cases} \sum_{e\in B} x_e \geq 1 \\ x_e \geq 0 \end{cases}$$

has integer vertices.

Let λ be an optimal solution of the program; we are going to use an "uncrossing" trick to construct from λ another solution, also optimal, and such that the cuts for which $\lambda_B > 0$ don't cross each other.

Let $B_i = \overrightarrow{E}(S_i, \overline{S_i})$ for $i = 1, 2$ be two cuts of the graph. We say that B_1 and B_2 *cross* if $S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1$ and $V \setminus (S_1 \cap S_2)$ are all non-empty. Then $B_1' = \overrightarrow{E}(S_1 \cap S_2, V \setminus (S_1 \cap S_2))$ and $B_2' = \overrightarrow{E}(S_1 \cup S_2, V \setminus (S_1 \cup S_2))$ are both dicuts, and it can be seen that

$$\chi^{B_1'} + \chi^{B_2'} = \chi^{B_1} + \chi^{B_2}$$

(because, the cuts being directed, there cannot be edges between $S_1 \setminus S_2$ and $S_2 \setminus S_1$).

If λ is an optimal solution such that two cuts B_1 and B_2 cross, we can then replace it by $\overline{\lambda}$, defined in function of $\epsilon = \min(\lambda_{B_1}, \lambda_{B_2})$ by:

$$\begin{cases} \overline{\lambda}_{B_1} & = \lambda_{B_1} - \epsilon, & \overline{\lambda}_{B_2} & = \lambda_{B_2} - \epsilon, \\ \overline{\lambda}_{B_1'} & = \lambda_{B_1'} + \epsilon, & \overline{\lambda}_{B_2'}, & = \lambda_{B_2'} + \epsilon, \\ \overline{\lambda}_C & = \lambda_C & & \text{otherwise.} \end{cases}$$

We have:

$$\sum_B \overline{\lambda}_B = \sum_B \lambda_B$$

and no constraint is violated by $\overline{\lambda}$; thus $\overline{\lambda}$ is a new optimal solution. Unfortunately, it does not necessarily have fewer crossings than the previous one; however, we can repeat the procedure until no crossings are left; the termination is guaranteed by checking that $\sum_B \lambda_B |S_B|^2$ strictly decreases (and ϵ never goes below the reciprocal of the least common denominator of the original solution).

(In fact, it can be shown that if the pair to be uncrossed is chosen appropriately, the procedure takes polynomial time, independently of the values of the original λ, assuming that at the start it has a polynomial number

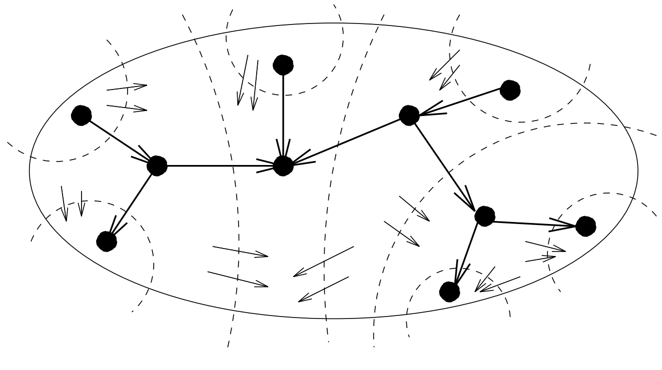

Figure 35: A nested family of cuts

of non-zero values. This is true if we obtain our starting optimal dual solution from the simplex algorithm: each non-zero component corresponds to independant constraints, and there can be at most as many as there are edges. But in this case, we will not need to follow the proof by an algorithm, so we don't care how many steps the uncrossing takes.)

So we may now assume that the cust B with $\lambda_B > 0$ are non-crossing.

These cuts are well structured: they can be described by a *nested family*, as we will now see. Take a reference graph vertex r. For each cut B, let $T(B)$ be the side of B which does not contain r. Thus $B = \overrightarrow{E}(T(B), \overline{T}(B))$ or $B = \overleftarrow{E}(T(B), \overline{T}(B))$ (see figure 35). The sets $T(B)$ associated with cuts with positive λ_B form a nested family:

$$\lambda_{B_1}\lambda_{B_2} > 0 \Longrightarrow T(B_1) \subset T(B_2) \text{ or } T(B_2) \subset T(B_1) \text{ or } T(B_1) \cap T(B_2) = \emptyset \ .$$

The graph is partitioned into regions by the cuts. For each B, take a representative node in the region $T(B) \setminus \cup_{T(B') \subset T(B)} T(B')$. (If the region is empty then create an artificial extra node to serve as a representative). We also consider r as a representative. Connect by an edge any two representatives which are separated by exactly one cut B.

We claim that this yields a tree on the representatives. If we direct the edges away from r, as we go along a path the number of cuts separating from r increases, thus we can never have a directed cycle; we obviously reach all the representatives from r, and each point has indegree exactly 1; thus we have a tree. Connecting all the remaining vertices of the original graph to the tree (the non-representative vertices of $T(B) \setminus \cup_{T(B') \subset T(B)} T(B')$ are connected to the representative) extends the tree to a spanning tree of G (the artificial vertices we now include in G).

Every dicut B such that $\lambda_B > 0$ is defined by an edge e of the tree (deleting it the tree splits into two, which defines a partition which defines B). (The tree may have edges which correspond to cuts B such that $\lambda_B = 0$.)

We now concentrate our attention on the cuts in

$$\mathcal{H} = \{B : \ \lambda_B > 0\}.$$

We want to solve the modified linear program

$$\begin{cases} \max \sum_{\mathcal{H}} y_B \\ y_B \geq 0 \\ \sum_{B \in \mathcal{H}, B \ni e} y_B \leq c(e) \end{cases}$$

The maximum in this problem will be the same as for the original problem. We just need to show that this new problem has an integral optimum solution.

As it turns out, this new problem is considerably simpler: it is totally unimodular: every submatrix has determinant 1, -1 or 0; thus a vertex, which is found by making some inequalities equalities and applying Cramer's rule, has integer coordinates (see class 1).

We use the matrix defined in the following way: G being a digraph and T a spanning tree of G, the matrix has $|E(T)|$ rows and $|E(G)| - |E(T)|$ columns. On the row corresponding to edge e, we put a 0 for each column corresponding to an edge which does not go across the cut defined by $T - e$, a 1 for each column corresponding to an edge which is correctly oriented (i.e. in the same direction as e), and a -1 for each edge which is incorrectly oriented. In the column of edge f, the non-zero entries then correspond to the edges of T which form a cycle with f, and their value is 1 or -1 depending on whether they follow the cycle in the direction opposite to f or not. The matrix thus defined is unimodular (this takes some effort to prove, but we skip the proof which is technical from our point of view; it is a good exercise if you want to understand the structure of matrices associated with graphs). Moreover, up to taking the transpose, multiplying some columns by -1 and taking a submatrix, we get the matrix of our modified linear program.

(There is an alternate, direct proof of the fact that the problem on $\mathcal{H}$ has an integral solution. Take y optimum solution, and suppose that there is a cut in $\mathcal{H}$ such that y_B is fractional. We can locally modify the fractional coordinates of y by $\pm\epsilon$ so that the constraints are still satisfied and the solution is still optimal.)

10.4 Reductions between minimization problems for clutters

$$\begin{tikzcd}
 & \textbf{Clutter} & \textbf{Blocker} & \text{LP} \arrow[d] \\
\text{greedy} \arrow[r] & s-t \text{ paths} \arrow[r, leftrightarrow] & s-t \text{ cuts} & \text{max flow} \arrow[l] \\
 & r\text{-branchings} \arrow[r, leftrightarrow] & r\text{-cuts} & \\
 & \text{dijoins} \arrow[r, leftrightarrow] & \text{dicuts} & \text{cuts} \arrow[l] \\
\text{weighted matchings} & t\text{-joins} \arrow[l] \arrow[r, leftrightarrow] & t\text{-cuts} &
\end{tikzcd}$$

11 Applications of Linear Programming to Matroids

11.1 Edmonds' theorem

The theorem of Edmonds gives linear inequalities defining the convex hull of incidence vectors of independent sets. Edmonds theorem, together with the duality theorem of linear programming implies

$$\max\{w(A) : A \in \mathcal{M}\} = \min\{\textstyle\sum_S y_S r(S) : y_S \geq 0, \sum_{S \ni e} y_S \geq w(e)\}$$

for $w : E \to \mathbf{R}$. The minimum program has exponentially many constraints. Later we will use this corollary of Edmonds' theorem to transform ugly minimum programs into easy-to-solve maximum programs.

Edmonds' theorem is:

Theorem 11.1 *If $(E, \mathcal{M})$ is a matroid, then*

$$\mathrm{conv}\{\chi^A : A \in \mathcal{M}\} = \{x \in \mathbf{R}^E : x \geq 0; \textstyle\sum_{e \in S} x_e \leq r(S), \forall S \subseteq E\}$$

where $r(S)$ is the rank of the subset S.

We define two polyhedra

$P_1 = \mathrm{conv}\{\chi^A : A \in \mathcal{M}\}$

$P_2 = \{x \in \mathbf{R}^E : x \geq 0; \sum_{e \in S} x_e \leq r(S), \forall S \subseteq E\}$.

We show $P_1 = P_2$.

The inclusion $P_1 \subseteq P_2$ is immediate since $\mathcal{M}$ is hereditary. (If $x = \chi^A$, then $\sum_{e \in S} x_e$ is the cardinality of $A \cap S$. Since $A \cap S$ is a subset of the independent set A, $A \cap S$ is independent. Hence the cardinality of the independent set $A \cap S$ in S is at most the rank of S.

There are two ways to establish the inclusion $P_1 \supseteq P_2$:

Method 1: Show that each vertex of P_2 equals χ^A for some $A \in \mathcal{M}$. (Note that P_2 is a bounded polyhedron, since for example no coordinte of a point in P_2 can exceed 1.)

Method 2: For each facet F of P_1 let H be the hyperplane containing F. Show that the halfspace defined by H which includes P_1 is described by one of the linear constraints defining P_2.

We now show how to prove Edmonds' theorem using each of these methods.

Proof using Method 1: If necessary, replace E by $\{e \in E : \exists A \in \mathcal{M}$ with $e \in A\}$. So we may assume P_1, hence P_2, has dimension $|E| = n$. Let $v \in \mathbf{R}^E$ be a vertex of P_2. Suppose the constraints which are satisfied exactly at this vertex are:

$$v_{e_1} = 0, \ldots, v_{e_T} = 0$$

$$v(S_1) = r(S_1), \ldots, v(S_M) = r(S_M)$$

where for $x \in \mathbf{R}^E$ and $S \subseteq E$ we use the notation $x(S) = \sum_{e \in S} x_e$. There are dim $P_2 = n$ of these equations which are linearly independent. Say

$$v_{e_1} = 0, \ldots, v_{e_t} = 0$$

$$v(S_1) = r(S_1), \ldots, v(S_m) = r(S_m)$$

where $t + m = n$. Without loss of generality, $t = 0$. (Otherwise delete $e_1, \ldots, e_t$ from E, and we're done by induction on $|E|$.) That is, $\chi^{S_1}, \ldots, \chi^{S_n}$ are linearly independent vectors of $\mathbf{R}^E$. To find a set $A \in \mathcal{M}$ with $v = \chi^A$ we must uncross the family $S_1, \ldots, S_n$.

Uncrossing involves repetitions of the following step: Take two of the sets, say S_1 and S_2. Replace one of them, say S_1, by $S_1 \cap S_2$ or $S_1 \cup S_2$. First we show that any such step yields a tight family. (Here the set S is tight if $v(S) = r(S)$.)

Claim: If S_1 and S_2 are tight, then $S_1 \cap S_2$ and $S_1 \cup S_2$ are tight.

Proof: Consider the string of inequalities:

$$\begin{array}{rcll} r(S_1) + r(S_2) & = & v(S_1) + v(S_2) & \text{since } S_1 \text{ and } S_2 \text{ aretight} \\ & = & v(S_1 \cap S_2) + v(S_1 \cup S_2) & \\ & \leq & r(S_1 \cap S_2) + r(S_1 \cup S_2) & \text{since } v \in P_2 \\ & \leq & r(S_1) + r(S_2) & \text{since } \mathcal{M} \text{ is a matroid} \end{array}$$

Second we show that careful choice of one of $S_1 \cap S_2$ or $S_1 \cup S_2$ yields a linearly independent family.

Claim: Suppose $S_1 \not\subseteq S_2$ and $S_1 \not\supseteq S_2$. If $\chi^{S_1}, \ldots, \chi^{S_n}$ is linearly independent, then either

$$\chi^{S_1 \cap S_2}, \chi^{S_2}, \ldots, \chi^{S_n} \text{ or } \chi^{S_1 \cup S_2}, \chi^{S_2}, \ldots, \chi^{S_n}$$

is linearly independent.

Proof: Notice

$$\chi^{S_1} + \chi^{S_2} = \chi^{S_1 \cap S_2} + \chi^{S_1 \cup S_2}.$$

So if $\chi^{S_1 \cap S_2}$ is linearly dependent on $\chi^{S_2}, \ldots, \chi^{S_n}$ and $\chi^{S_1 \cup S_2}$ is linearly dependent on $\chi^{S_2}, \ldots, \chi^{S_n}$, then χ^{S_1} is linearly dependent on $\chi^{S_2}, \ldots, \chi^{S_n}$.

Using these two claims, we show that careful uncrossing yields a chain in a finite number of steps.

The careful uncrossing proceeds by stages which modify the family. At the end of each stage either the maximum cardinality of a bad set decreases (a set S_i is bad if there is a set S_j with $S_i \not\subseteq S_j$ and $S_i \not\supseteq S_j$) or the maximum cardinality of a bad set remains the same and the number of bads sets attaining this maximum decreases. Hence there are a finite number of stages.

Say at the beginning of a stage of the uncrossing the family is $S_1, \ldots, S_n$. Relabel so that S_1 is the largest of the bad sets in the family. Relabel so that S_2 is a set that makes S_1 bad. ($S_1 \not\subseteq S_2$ and $S_1 \not\supseteq S_2$) Depending on the choice that preserves linear independence, we replace S_1 by $S_1 \cap S_2$ or $S_1 \cup S_2$. In the first case, the stage ends. In the second case, if $S_1 \cup S_2$ is good then the stage ends. In the second case, if $S_1 \cup S_2$ is bad then relabel so S_3 is a set that makes $S_1 \cup S_2$ bad. Again depending on the choice that preserves linear independence, we replace $S_1 \cup S_2$ by $(S_1 \cup S_2) \cap S_3$ or by $S_1 \cup S_2 \cup S_3$. Again in the first case the stage ends. In the second case, if $\cup_i S_i$ is good the stage ends. Otherwise find S_{i+1} which makes it bad. Hence the stage ends in at most $n-1$ steps.

After careful uncrossing, our modified family is a chain $S_1 \subseteq \cdots \subseteq S_n$. Since $\chi^{S_1}, \ldots, \chi^{S_n}$ is linearly independent, $S_1 \neq \emptyset$ and $S_i \neq S_j$ for $i \neq j$. So $S_1 \subset \cdots \subset S_n$. Therefore $|S_i| = i$ and with an appropriate numbering of the elements of E,

$$S_i = \{e_1, \cdots, e_i\}.$$

Now $v(S_i) = r(S_i)$, so

$$\begin{aligned} v_{e_1} &= r(S_1) \\ v_{e_2} &= r(S_2) - r(S_1) \\ &\vdots \\ v_{e_n} &= r(S_n) - r(S_{n-1}) \end{aligned}$$

Therefore, v is an integral vector. Hence, as can be seen directly from the constraints $x_e \leq r(\{e\})$ defining P_2, v is a 0-1 vector. Say $v = \chi^A$ for $A \subseteq E$. Notice $r(A) = |A|$ since $|A| = v(A) \leq r(A) \leq |A|$. (The inequality $v(A) \leq r(A)$ follows holds since v is in P_2.) Hence $A \in \mathcal{M}$. This completes the proof of the theorem using Method 1.

Proof using Method 2: This method uses the fact that the greedy algorithm yields an optimal solution to the linear program $\max\{w(A) : A \in \mathcal{M}\}$, when $\mathcal{M}$ is the family of independent sets of a matroid $(E, \mathcal{M})$ and $w : E \to \mathbf{R}_+$.

Let F be a facet of P_1 contained in a hyperplane $\sum a_e x_e = \alpha$, where P_1 lies in the halfspace $\sum a_e x_e \leq \alpha$. We would like to show that $\sum a_e x_e \leq \alpha$ is one of the constraints defining P_2.

Case 1: There is $e_0 \in E$ with $a_{e_0} < 0$.

In this case, $v_{e_0} = 0$ for all vertices v on the facet F of P_1. (If $v_{e_0} = 1$ for some vertex $v = \chi^A$ on F, then $\sum_{e \neq e_0} a_e v_e = \alpha - a_{e_0} v_{e_0} > \alpha$, so $\chi^{A - \{e_0\}}$ is not is P_1.) So the facet F is in the hyperplane $x_{e_0} = 0$.

Case 2: For all $e \in E$, $a_e \geq 0$.

Label the elements of E so that $a_{e_1} = a_{e_2} = \cdots = a_{e_k} > a_{e_{k+1}} \geq \cdots \geq 0$. Since $P_1 = \operatorname{conv}\{\chi^S : S \in \mathcal{M}\}$,

$$\alpha = \max\{\textstyle\sum_{e \in S} a_e : S \in \mathcal{M}\}.$$

We can compute $\max\{\sum_{e \in S} a_e : S \in \mathcal{M}\}$ by the greedy algorithm. Add to S first e_1, then perhaps e_2, …, skipping those elements which would cause S to be dependent. Say $\{e_{i_1}, \ldots, e_{i_r}\} = S \in \mathcal{M}$ is produced by the greedy algorithm, with $i_1 < i_2 < \cdots < i_r$. For k defined above, suppose exactly s of these indices do not exceed k. Then the greedy algorithm yields $r(\{e_1,\dots,e_k\}) = s$. Let χ^A be any vertex on F with $\sum_{e\in A} a_e = \alpha$. Then

Claim: The independent set A contains exactly s elements from $e_1,\dots,e_k$.

Proof: If not, then $B = A \cap \{e_1,\dots,e_k\}$ is not a basis of $\{e_1,\dots,e_k\}$. But B is an independent set in $\{e_1,\dots,e_k\}$ so there is some $e_j \notin B$ with $B \cup \{e_j\}$ still independent and in $\{e_1,\dots,e_k\}$. By the exchange property there is $e_\ell \in A$ with $\ell > k$, such that $(A - \{e_\ell\}) \cup \{e_j\}$ is an independent set. Call this set C. Then

$$\sum_{e\in C} a_e = \sum_{e\in A} a_e + a_{e_j} - a_{e_\ell} > \alpha.$$

since $j < \ell$. This contradicts the fact that $\chi^C \in P_1$. The claim is established.

We have shown for all A such that χ^A is a vertex of the facet lying in $\sum a_e x_e = \alpha$ that

$$|A \cap \{e_1,\dots,e_k\}| = r(\{e_1,\dots,e_k\}).$$

So all points x on the facet satisfy

$$x_{e_1} + \cdots + x_{e_k} = r(\{e_1,\dots,e_k\}).$$

So the halfspace $\sum a_e x_e \le \alpha$ is described by one of the constraints defining P_2.

11.2 The ellipsoid method applied to matroid intersection

We now present the polyhedral description of matroid intersection.

Theorem 11.2 *Let* $(E, \mathcal{M}_i)$ *be two matroids* ($i = 1,2$)*. Then*

$$Conv\{\chi^A, A \in \mathcal{M}_1 \cap \mathcal{M}_2\} = \{x \in \mathcal{R}^E, x \ge 0, \forall S \;\; x(S) \le \min(r_1(S), r_2(S))\}\,.$$

Thus the polyhedron of the matroid intersection is exactly the intersection of the two matroid polyhedra. The $\subset$ direction is of course obvious. The other direction can be proved using uncrossing, as in the proof of last class (only more complicated).

Since the system of equations is totally dual integral, this theorem implies polynomial time optimization. For that, we need a converse to our main result on the ellipsoid method. We have seen that if K is a polyhedron for which we have a separation oracle (and we know a bound on the complexity of the linear inequalities defining K), then we can optimize any linear objective function over K in polynomial time. Let us show that the converse is also true:

Theorem 11.3 *Let K be a polyhedron in $\mathcal{R}^n$. Suppose that we know $\phi > 0$ such that K can be defined by inequalities of the form $\sum a_i x_i \leq \alpha$, where $|a_i| \leq 2^\phi$, $|\alpha| \leq 2^\phi$, and a_i and α are integers. Assume that we have an oracle to maximize any linear objective function over K. Then the separation problem for K can be solved in time poly($\phi, n, \#$digits in c).*

Checking whether $y \notin K$ means finding w such that $\max_{x \in K} w \cdot x < w \cdot y$. Assume for simplicity that 0 is in the interior of K; then (by normalizing) this is equivalent to finding a vector w such that $w \cdot x \leq 1$ for $x \in K$ and $w \cdot y > 1$. Thus, if we consider the polyhedron

$$W = \{w : \forall x \in K,\ w \cdot x \leq 1\} ,$$

we want to maximize $w \cdot y$ over W. Since separation can be checked easily for W (because we can optimize over K by hypothesis), and bounds on the coefficients of a linear description of W can also be computed in polynomial time, we can optimize over W by our general theorem. Hence we can separate from K in polynomial time.

Now let $P_i = \text{Conv}\{\chi^A, A \in \mathcal{M}_i\}$. We know that we can optimize over P_i by the greedy algorithm. So we can solve the separation problem for P_i in polynomial time. Hence it is trivial to solve the separation problem for $P_1 \cap P_2$ (just apply the separation algorithm for P_1 and P_2 separately). Thus we can solve the optimization problem for $P_1 \cap P_2$ in polynomial time.

There is a direct combinatorial (alternating path type) algorithm to solve the weighted matroid intersection problem, so the use of the ellipsoid method is not crucial here.

We have also solved the separation problem for a matroid polytope, i.e. the problem: Given a matroid $(E, \mathcal{M})$ and a vector $x \in R^E$, decide if x is a convex combination of incidence vectors of independent sets, and return a set $S \subset E$ with $x(S) > r(S)$ if not. For this problem there is now a combinatorial polynomial algorithm by Cunningham, but the first polynomial time method was found with the ellipsoid method.