NOTES ON COMBINATORIAL OPTIMIZATION

GEIR DAHL*
and
CARLO MANNINO†

October 1, 2012

* Department of Mathematics and Department of Informatics, CMA, University of Oslo, Norway. geird@math.uio.no
† University of Rome, La Sapienza, Department of Computer Science, Rome, Italy, and visiting scientist at CMA. mannino@dis.uniroma1.it

Contents

List of Figures

Chapter 1

Combinatorial optimization and polyhedral combinatorics

Consider the following *combinatorial optimization problem* (CO). We have a class $\mathcal{F}$ of *feasible sets* each being a subset of a finite ground set E; we are also given a weight function $w : E \to \mathbb{R}$, and for $F \in \mathcal{F}$ we define its weight $w(F) = \sum_{e \in F} w(e)$. Then the combinatorial optimization problem is:

Combinatorial Optimization Problem

$$\max\{w(F) : F \in \mathcal{F}\} \tag{1.1}$$

One major goal in the area *polyhedral combinatorics* is to translate the (CO) problem into an optimization problem in $\mathbb{R}^E$. To this end, we represent each $F \in \mathcal{F}$ by its incidence vector χ^F in $\mathbb{R}^E$; so $\chi^F_e = 1$ if $e \in F$ and $\chi^F_e = 0$ otherwise. Let $S = \{\chi^F : F \in \mathcal{F}\} \subseteq \{0,1\}^E$ be the set of the incidence vectors of the sets in $\mathcal{F}$. Then the following problem is equivalent to (CO):

$$\max\{w^T x : x \in S\} \tag{1.2}$$

project selection problem

Example 1.1. Project Selection. *An organization has stated that the budget for all projects for the upcoming half-year is \$600,000, which must cover all internal and external costs associated with the projects. In addition, the amount of work that can be accomplished by the organization's personnel is limited to 5,000 hours for the half-year. The project manager has picked up two possible project alternatives, say A1 and A2. Project A1 gives a profit of \$400,000 and requires an investment of \$300,000 plus 4000 hours of personnel. Project A2 gives a profit of \$ 500,000 and requires an investment of \$400,000 plus 2000 hours of personnel. We need to find an investment which maximizes profit, satisfying the budget and personnel constraints.*

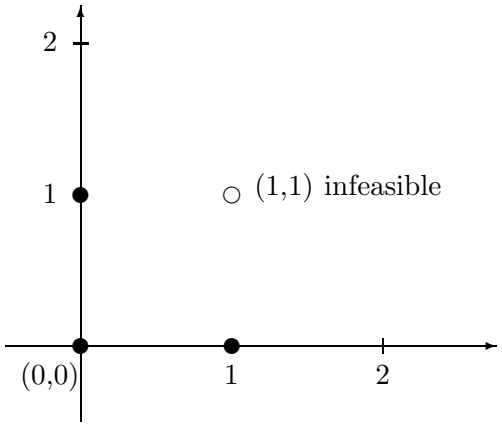

Figure 1.1: The feasible points (solid circles) of Example 1.1

The example can immediately be cast into the (CO) paradigm. So, the ground set is $E = \{A1, A2\}$, with $w_1 = 400000$ and $w_2 = 500000$. Since we cannot activate both projects without violating the constraints, the family of feasible solutions is $\mathcal{F} = \{F_1, F_2, F_3\}$, where $F_1 = \{A1\}$ corresponds to select only the first alternative, $F_2 = \{A2\}$ corresponds to the second alternative, while $F_3 = \emptyset$ is the *no investment* alternative. The associated set of feasible 0-1 solutions is then $S = \{(1, 0), (0, 1), (0, 0)\}$. We represent S geometrically in the plane in Figure 1.

travelling salesman problem

Example 1.2. *The Travelling Salesman Problem (TSP) is to find a shortest trip (called a* tour*) through a given set of cities. More precisely, we are given an undirected graph $G = (V, E)$ with nonnegative weights (costs) on the edges: w_e, $e \in E$. A* tour *is a Hamilton cycle, i.e., a cycle going through all the nodes of G and passing through each node exactly once. The length of a tour is the sum of the weights of its edges. The problem is to find a shortest tour.*

In this example, the ground set coincides with the set E of edges of G; the feasible solutions (sets) are the edge sets $F \subseteq E$ corresponding to the Hamilton cycles of G.

The general combinatorial optimization problem belongs to the class of NP-hard problems. We cannot give here a formal definition of such a class. It is enough to observe that for most problems in this class we do not have efficient solution algorithms, that is, algorithms which are able to find an optimal solution within a reasonable amount of time. More precisely, the time spent by the best known solution algorithm can grow exponentially in $|E|$.

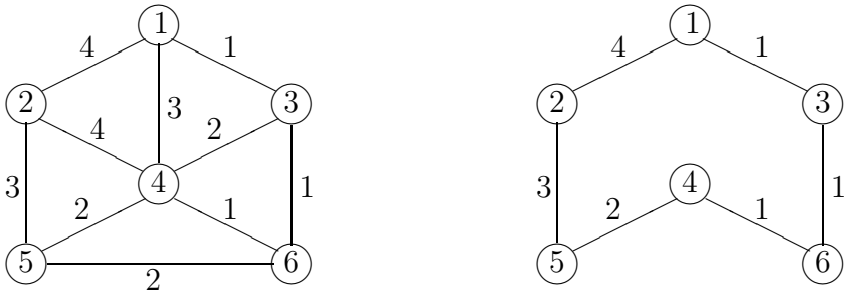

Figure 1.2: An example of Hamilton tour

The area of *polyhedral combinatorics* attempts to solve (1.2) (and thus (CO)) by means of linear programming. This is done by exploiting the properties of the convex hull $\text{conv}(S)$ of S, which is a polytope[1]. If we denote by $\text{ext}(P)$ the set of vertices of a polyhedron P, we have $S = \text{ext}(\text{conv}(S))$. Consider now the following linear programming problem:

$$\max\{w^T x : x \in \text{conv}(S)\} \tag{1.3}$$

If S is empty, so it is $\text{conv}(S)$. Otherwise $\text{conv}(S) \neq \emptyset$ and there exists an optimal solution $x^* \in \text{ext}(\text{conv}(S)) = S$. So, in principle (CO) can be solved by applying the simplex method to (1.3), if we are given an external representation of $\text{conv}(S)$ in terms of a system of linear inequalities. Unfortunately, for the great majority of combinatorial optimizations, such a representation is not at hand, not even implicitly.

There exist, however, some remarkable exceptions. One such exception is the problem of finding a *maximum weight forest* in a given connected, undirected graph $G = (V, E)$ with weight function w; this example is from [18]. Recall that a forest is an acyclic undirected graph (whereas a tree is a connected forest). The weight of a forest is simply the sum of weights of its edges, which implies that when looking for maximum weight forests we are not interested in including isolated nodes[2]. So, it is natural to identify a forest $H = (V', F)$ of $G = (V, E)$ with the set F of its edges. Note also that if all weights are nonnegative the maximum weight forest is equivalent to the spanning tree problem (show this!).

maximum weight forest problem

With negative weights present, an optimal solution may be a forest and not a

1. Recall that a polytope P is the convex hull of a finite set of points in $\mathbb{R}^n$, and, by the main representation theorem for polyhedra, a set is a polytope if and only if it is a bounded polyhedron.
2. A node is *isolated* in a graph H if no arcs of H are incident to it.

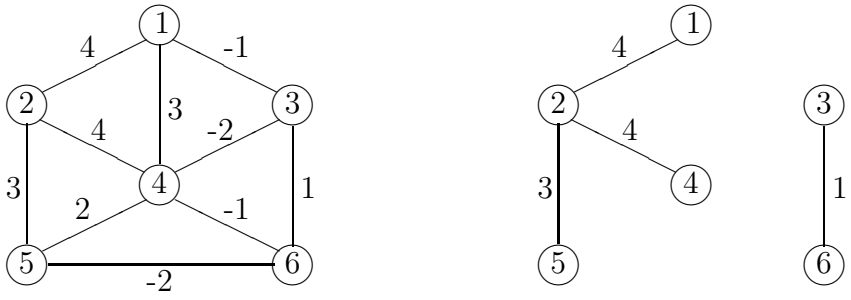

Figure 1.3: An example of maximum weight forest

tree. Let $G = (V, E)$ be a graph and, for each $S \subseteq V$, denote by $E(S)$ the set of edges with both ends in S. It is easy to show that G is a forest if and only if, for each $S \subseteq V$, we have $|E(S)| \leq |S| - 1$. This translates immediately into the following characterization of the incidence vectors of forests:

Lemma 1.1. *Let $G = (V, E)$ be a graph. Then $x \in \{0, 1\}^E$ is the incidence vector of (the edges of) a forest if and only if*

$$x(E[S]) \leq |S| - 1 \quad \text{for all } S \subseteq V. \tag{1.4}$$

Inequalities of type (1.4) are called *subtour elimination constraints*.

In Chapter 1 of [21] the spanning tree problem is solved in polynomial time by the greedy algorithm. The idea is to iteratively extend a forest by adding a minimum weight edge which does not introduce any cycles. This algorithm also solves the maximum weight forest problem, provided that we terminate whenever the next edge has nonpositive weight.

First we define the polytope of interest. We let the *forest polytope* $F(G)$ be the convex hull of the incidence vectors of all forests in G. Jack Edmonds showed in [7] the following theorem which gives a complete linear description of $F(G)$.

Theorem 1.2. *$F(G) \subset \mathbb{R}^E$ is the solution set of the following linear system*

$$\begin{array}{lll} \text{(i)} & x_e \geq 0 & \text{for all } e \in E; \\ \text{(ii)} & x(E[S]) \leq |S| - 1 & \text{for all } S \subseteq V. \end{array} \tag{1.5}$$

Proof. We follow the presentation of [18]. The idea of this proof, due to Edmonds, is to show that every vertex of the bounded polyhedron defined by (1.5) is the incidence vector of a forest of G. This in turn is done by using the greedy algorithm on the LP dual of the problem of maximizing $c^T x$ subject to the linear system (1.5). Also, via complementary slackness, an optimal primal solution is constructed.

Let Q denote the polytope being the solution set of (1.5), and we shall prove that $Q = F(G)$. Note that the integral vectors in Q are precisely the incidence vectors of forests by Lemma 1.1. By convexity this implies that $F(G) \subseteq Q$.

To prove the reverse inclusion, let $\bar{x}$ be a vertex of Q, i.e., this point is the *unique* optimal solution an LP problem $\max\{c^T x : x \in Q\}$ for suitable $c \in \mathbb{R}^n$. The LP dual of this problem is

$$\begin{array}{lll} \min & \sum_{S \subseteq V} y_S(|S| - 1) & \\ \text{subject to} & & \\ \text{(i)} & \sum_{S: e \in E[S]} y_S \geq c_e; & \text{for all } e \in E; \\ \text{(ii)} & y_S \geq 0 & \text{for all } S \subseteq V. \end{array} \tag{1.6}$$

Consider the greedy algorithm applied to the primal problem, and assume that the edges found are $F = \{e_1, \ldots, e_s\}$ in that order. Thus, in the i'th iteration the edge e_i is added and it joins two components of the current solution and forms the new component V_i. (For instance, when $i = 1$ we have n components and $e_i = [u, v]$ joins the components $\{u\}$ and $\{v\}$, so $V_1 = \{u, v\}$.)

The dual solution y is defined next. We let $y_S = 0$ for all sets S not among the sets V_i, $i = 1, \ldots, s$. The values for the sets V_i are determined in the reverse order as follows. Let $y_{V_r} = c(e_r)$. Consider the primal greedy solution x' found above. The complementary slackness condition for edge e says, as $x'_{e_r} > 0$, that $\sum_{S: e_r \in E[S]} y_S = c(e_r)$. With our definition of y_{V_r} this equality already holds, and we are forced to define $y_S = 0$ for all other sets S that contain both endnodes of e_r. To define the remaining components of y, let, for each $j \in \{1, \ldots, r-1\}$, $I(j) = \{i : j + 1 \leq i \leq r \text{ and both end nodes of } e_j \text{ are in } V_i\}$ and define $y_{V_j} = c(e_j) - \sum_{i \in I(j)} y_{V_i}$ for $j = r-1, r-2, \ldots, 1$.

One can now check that y is dual feasible, x' is primal feasible and that the complementary slackness condition holds. Thus it follows from LP theory that both these solutions are optimal in the respective problems. But $\bar{x}$ was also optimal in the primal, and due to the uniqueness, we must have $\bar{x} = x'$, which proves that every vertex of Q is integral and we have $Q = F(G)$ as desired.

□

In most cases, an external representation of $\text{conv}(S)$ as the solution set of a system $Ax \leq b$ is not available. However, we will show in the next section that it may be very useful to have at hand some suitable relaxations of $\text{conv}(S)$, denoted as *formulations* (of S).

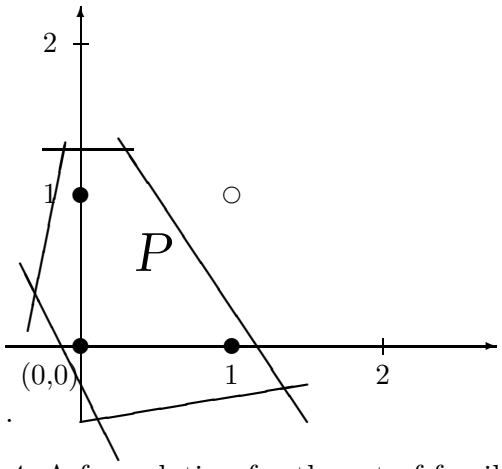

Figure 1.4: A formulation for the set of feasible solutions of Example 1.1

1.1 Formulations and relaxations for (CO)

We start with an important definition.

Definition 1.1. *Let $S \subseteq \{0,1\}^n$. A* formulation *for S is a polyhedron $P \subseteq \mathbb{R}^n$ satisfying $P \cap \{0,1\}^n = S$.*

In other words, P is a formulation for S iff the 0-1 points of P are precisely the points of S. A possible formulation for the set of feasible solutions of Example 1.1 is given in Figure 1.1. Clearly, one can build an infinite number of different formulations for the same set S. Since $\text{conv}(S) \cap \{0,1\}^n = S$, it follows that $\text{conv}(S)$ is one such formulation for S. A special role is played by the so called *natural formulation*, which is the one directly obtained from the constraints of the original problem when this is stated as a linear pogramming problem with integrality constraints on the variables. In particular, S is sometimes directly defined as the set of binary solutions to a family of linear constraints. In other cases, such a family of linear constraints may be derived directly from the description of the problem, as in Example 1.1. Of course, a natural formulation may not be available in general.

natural formulation

Example 1.1 *(continued).* Let S be the set of feasible solutions of Example 1.1, and let $x \in S$. Since x is a binary vector, then it satisfies:

$$0 \leq x_i \leq 1 \qquad i = 1, 2 \tag{1.7}$$

Also, x satisfies the budget constraint:

$$300000x_1 + 400000x_2 \leq 600000 \tag{1.8}$$

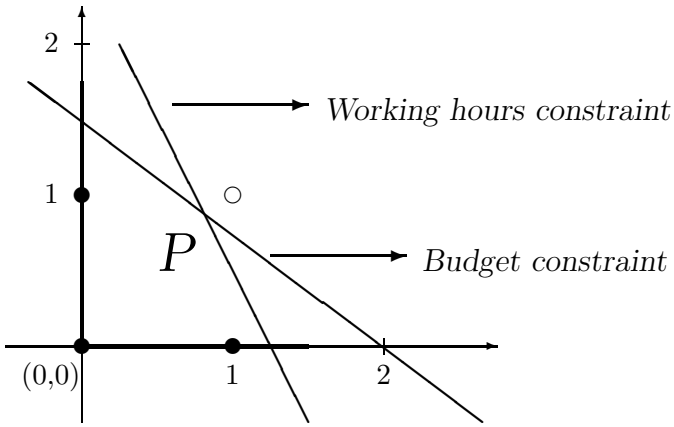

Figure 1.5: The natural formulation for the set of solutions of Example 1.1

and the working hours constraint:

$$5000x_1 + 2000x_2 \leq 6000 \tag{1.9}$$

Constraints (1.7), (1.8) and (1.9) define a natural formulation for S (see Picture 1.1).

Example 1.2 *(continued).* Consider again the TSP problem associated with a graph $G = (V, E)$. One can show that a 0-1 vector x is the incidence vector χ^F of a Hamilton tour $F \subseteq E$ iff it satisfies the following system of linear inequalities

$$\begin{array}{lll} \text{(i)} & x(\delta(v)) = 2 & \text{for all } v \in V; \\ \text{(ii)} & x(\delta(W)) \geq 2; & \text{for all } W \subset V,\ W \neq \emptyset,\ W \neq V. \end{array} \tag{1.10}$$

The constraints (i) assure that every 0-1 solution x is of the form $x = \chi^F$ where $F \subseteq E$ and $d_{(V,F)}(v) = 2$ for each $v \in V$; such a set F is called a $2 - matching$ (or 2-factor). Clearly, every tour is a 2-matching, so all these inequalities must be satisfied by the incidence vectors of Hamilton tours. However, in general a 2-matching is a union of disjoint cycles, so it may not be a tour. The *2-connectivity inequalities* (ii) eliminate such a possibility, i.e., a 2-matching that satisfies (ii) is a tour (otherwise we could let W be the node set of one subtour, and we would get a violated inequality). From this it is not difficult to see that the 0-1 feasible solutions in (1.10) are precisely the incidence vectors of tours.

An equivalent set of constraints that can replace (1.10)(ii) is the set of *subtour*

elimination constraints:

$$x(E[S]) \leq |S| - 1 \text{ for all } S \subseteq V,\ S \neq \emptyset,\ S \neq V.$$

These constraints were introduced in the 1954 paper by Dantzig, Fulkerson and Johnson [5]. Note that the number of 2-connectivity inequalities, or subtour elimination constraints, grows exponentially as a function of the number of nodes.

Relaxations.

In what follows we will often deal with mathematical optimization problems of the form $\max\{f(x) : x \in S \subseteq \mathbb{R}^n\}$, and with their optimal values. Conventionally, we will denote such a problem by a capital roman letter between brackets, e.g. (Q); also, we denote its value by $v(\mathrm{Q}) = \max\{f(x) : x \in S \subseteq \mathbb{R}^n\}$ where we permit the values $v(\mathrm{Q}) = -\infty$ (when the problem is infeasible) and $v(\mathrm{Q}) = \infty$ (when the problem is unbounded).

Now, suppose we want to solve the combinatorial optimization $\max\{w^T x : x \in S\}$ (where $S \subseteq \{0,1\}^n\}$) denoted by (Q), and assume we have at hand a feasible solution $\bar{x} \in S$ to (Q). How good is $\bar{x}$? In other words, how does the value $w^T\bar{x}$ of $\bar{x}$ compare with the optimal value $v(\mathrm{Q})$ to (Q)? We will see that formulations can provide such an answer. First we need to introduce the concept of a relaxation of an optimization problem.

Definition 1.2. *Let* $S \subseteq \mathbb{R}^n$ *and let* (Q) *be* $\max\{w(x) : x \in S\}$. *The problem* (R) $\max\{g(x) : x \in T\}$ *is a* relaxation *of* (Q) *iff* $T \supseteq S$ *and* $g(x) \geq w(x)$ *for all* $x \in S$.

The following theorem is a straightforward consequence of the above definition:

Theorem 1.3. *Let* (Q) *be a maximization problem (as above) and let* (R) *be a relaxation of* (Q). *Then* $v(R) \geq v(Q)$.

relaxation quality

In other words, the optimal value of (R) provides an *upper bound* (UB) on the optimal value of (Q). Let's go back to our original question. In many cases, it may be very difficult to find an optimal solution x^* of an optimization problem Q, whereas it can be easy to obtain a feasible solution $\bar{x} \in S$. This is often the case when (Q) is a combinatorial optimization. Clearly, $w(\bar{x}) \leq w(x^*) = v(\mathrm{Q})$, and $w(\bar{x})$ is a *lower bound* (LB) on the optimal value to (Q). Suppose now that we have at hand a relaxation (R) $\{\max g(x) : x \in T\}$ of (Q) and assume that an optimal solution y^* to R can be computed efficiently. We have

$$UB = v(\mathrm{R}) = g(y^*) \geq w(x^*) \geq w(\bar{x}) = LB.$$

The optimal value of (Q) belongs to the interval between the lower and the upper bound (we are assuming that both can be computed efficiently). Its size $UB - LB$ is called the *gap*. The smaller the gap is, the better it is. If $gap = 0$ then $UB = LB$, which in turn implies $UB = v(\mathrm{R}) = g(y^*) = w(x^*) = w(\bar{x}) = LB$, which proves that $\bar{x}$ is an optimal solution to (Q). So, in general, we would like the upper bound to be as small as possible; in contrast, the lower bound should be as large as possible. In other words, we want to have at hand

- *strong relaxations* of (Q), providing small upper bounds on $v(\mathrm{Q})$ and
- effective *heuristic methods*, capable to find good quality solutions to (Q), hopefully optimal.

The rest of this chapter is devoted to the first goal, whereas heuristic methods will be discussed in a subsequent chapter.

Now, let (Q) be $\max\{w^T x : x \in S\}$ where $S \subseteq \{0,1\}^n$ be a combinatorial optimization problem, and let $P \subseteq \mathbb{R}^n$ be a formulation for S. By definition, $S \subseteq P$. It follows from the above discussion that the following problem

$$\text{(R)} \qquad \max\{w^T x : x \in P\}$$

is a relaxation of (Q). Since problem (R) amounts to maximizing a linear function over a polyhedron, (R) is a linear program (LP), and we there are a number of efficient algorithms to solve LPs. The simplex method is one of these, even though its theoretical complexity is still unknown. This means that the simplex method works very well in practice, but in principle there exist highly complicated instances requiring an enormous amount of time to be solved. Interestingly, theoretically more efficient methods as the *interior point algorithms* do not perform significantly better in practice. So, problem (R) along with the simplex method can be used to compute upper bounds on $v(\mathrm{Q})$.

comparing formulations

As already mentioned, we may have different formulations for the same set S. How can we distinguish among them? We need some criteria to compare between formulations and to measure their quality. So, let (Q) be $\max\{w^T x : x \in S \subseteq \{0,1\}^n\}$ be a combinatorial optimization, and let P_1 and P_2 be two distinct formulations for S. Since we want to use P_1 and P_2 to compute upper bounds on $v(\mathrm{Q})$, one candidate quality criterium is the following: P_1 is *better* than P_2 if and only if $\max\{w^T x : x \in P_1\} \leq \max\{w^T x : x \in P_2\}$. So, if we consider the two formulations in Figure 1.1 for the problem of Example 1.1, the natural formulation P_1 has to be preferred to the other when the objective function is the original one, to maximize $4x_1 + 5x_2$.

However, if the objective becomes $\max\ x_2$, then P_2 has to be preferred to P_1. So, this indicator is dependent on the objective and thus on the problem. It is

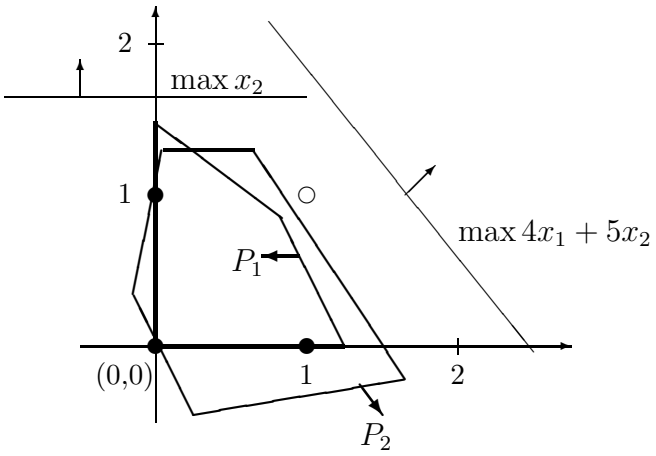

Figure 1.6: Comparisons among different formulations of Example 1.1

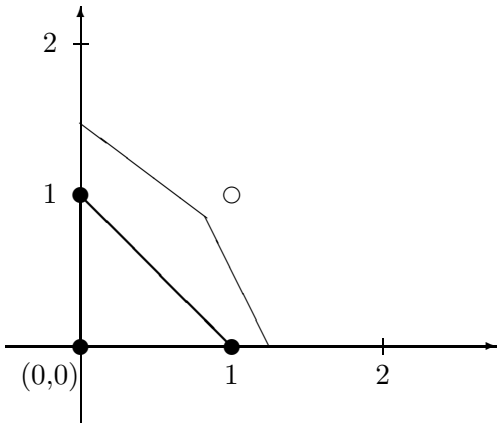

Figure 1.7: Best against natural formulation of Example 1.1

preferable to establish a criterium which only depends on the formulations. So, we say that P_1 is better than P_2 iff $P_1 \subseteq P_2$. This implies that $\max\{w^T x : x \in P_1\} \leq \max\{w^T x : x \in P_2\}$ for *any* $w \in \mathbb{R}^n$ and P_1 will always return a bound which is not worse than that associated with P_2. Actually, from convexity we have the following theorem:

Theorem 1.4. *Let P_1 and P_2 be polytopes in $\mathbb{R}^n$. Then $P_1 \subseteq P_2$ if and only if $\max\{w^T x : x \in P_1\} \leq \max\{w^T x : x \in P_2\}$ for all $w \in \mathbb{R}^n$.*

Now, it is not difficult to see that $\text{conv}(S)$ is contained in every formulation for S, and we say that $\text{conv}(S)$ is the *best formulation* for S. Indeed, we have that $\text{conv}(S)$ returns the best possible upper bound, since for all $w \in \mathbb{R}^n$, we have: $\max\{w^T x : x \in \text{conv}(S)\} = \max\{w^T x : x \in S\}$.

From the above discussion, it is desirable to have at hand an external representation of $\text{conv}(S)$, that is a coefficient matrix $A \in \mathbb{R}^{m\times n}$ and a vector $b \in \mathbb{R}^m$ such that $\text{conv}(S) = \{x \in \mathbb{R}^n : Ax \leq b\}$. Such a description does exist due to the Main Theorem of Polyhedra. Unfortunately, in many cases A and b are not known, not even implicitly (as for example for the set of the feasible solution to the Maximum Weight Forest Problem). In practice in most cases we can only find a (small) subset of such a linear system $Ax \leq b$, typically associated with some special structure of the coefficients.

Example 1.3. *Let* $S = \{x \in \{0,1\}^5 : 7x_1 + 6x_2 + 5x_3 + 3x_4 + 2x_5 \leq 11\}$. *Note that* S *is the set of binary vectors satisfying a single linear constraint, also called* knapsack constraint. *It can be shown that* $\text{conv}(S)$ *is contained in the following polytope:*

$$\begin{aligned}
&y_1 + y_2 \leq 1\\
&y_1 + y_3 \leq 1\\
&y_1 + y_4 + y_5 \leq 2\\
&y_1 + y_2 + y_3 + y_4 \leq 2\\
&y_1 + y_2 + y_3 + y_5 \leq 2\\
&3y_1 + 2y_2 + 2y_3 + y_4 + y_5 \leq 4\\
&0 \leq y_i \leq 1 \ \ (i \leq 5)
\end{aligned} \tag{1.11}$$

The last inequalities are box constraints, satisfied by every 0-1 vector. The first 5 constraints are facets of $\text{conv}(S)$ *and have special structure. In particular, each constraint corresponds to a subset* C *of variables whose coefficients in the knapsack constraint sum up to something strictly greater than 11. So, if* $x_i = 1$ *for all* $i \in C$, *the knapsack constraint is violated. Which implies that* $\sum_{i\in C} x_i < |C|$, *or, equivalently since we are interested in binary solutions,* $\sum_{i\in C} x_i \leq |C| - 1$.

In some special, well-behaved cases, we have at hand an explicit description of the convex hull of the feasible solutions. This is the case for the Maximum Weight Forest problem, whose convex hull is given by (1.5). However, the number of constraints (1.5.ii) is equal to the number of subsets of S which, in turn, grows exponentially in $|S|$. So, if $|S| = 101$, we have 2^{100} constraints of type (1.5.ii), and we may forget to list them all! Luckily, we can still solve the optimization problem $\max\{w^T x : Ax \leq b, x \in \mathbb{R}^n\}$ without generating explicitly all rows of A.

1.2 Separation oracles and the dynamic simplex method

Let (Q) be $\max\{w^T x : Ax \leq b,\ x \in \mathbb{R}^n\}$, with $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, be a linear programming problem. There are several interesting cases in which A and

b cannot be explicitly represented, typically because m is too large and such a representation would require too much computer space and time to be obtained. One such example is the formulation (1.5) of the Maximum Weight Forest in a graph G. The number of constraints of type (1.5.ii) grows exponentially in the number of nodes of G, which makes it impossible to generate all of them even for small graphs. However, this is not necessary to solve the corresponding optimization problem. To understand this, we need to introduce the concept of separation oracles.

Definition 1.3. *Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, let $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ be a polyhedron and let $\bar{x} \in \mathbb{R}^n$. A* separation oracle *is an algorithm which either concludes that $\bar{x} \in P$ or, if $\bar{x} \notin P$, returns a constraint $\sum_j a_{ij}x_j \leq b_i$ in the description of P violated by $\bar{x}$ (i.e. $\sum_j a_{ij}\bar{x}_j > b_i$).*

Example 1.3 *(continued).* Consider the formulation P for the set $S = \{x \in \{0,1\}^5 : 7x_1 + 6x_2 + 5x_3 + 3x_4 + 2x_5 \leq 11\}$ obtained by considering the knapsack constraint $7x_1 + 6x_2 + 5x_3 + 3x_4 + 2x_5 \leq 11$ and all constraints (1.11) and let $\bar{x}_1 = 2/3$, $\bar{x}_2 = 2/3$, $\bar{x}_3 = \bar{x}_4 = \bar{x}_5 = 0$. Then a separation oracle for P would return $\bar{x} \notin P$, along with the violated cover inequality $x_1 + x_2 \leq 1$. Remark that the knapsack constraint is not violated by $\bar{x}$.

In the above example the oracle finds a violated constraint by inspection, as the number of inequalities is very small and they are listed explicitly. This is in general not the case and separation oracles can be rather complicated algorithms, not necessarily efficient. We will describe now an efficient, LP-based algorithm to solve the separation problem for the forest polytope.

1.3 A separation oracle for the forest polytope.

Let $\bar{x} \in \mathbb{R}^E$: we need to establish whether $\bar{x}$ violates one of the constraints of (1.5) and, if so, which one. Clearly, if $\bar{x}_i < 0$ for some $i \in E$, then an oracle could check it by simple inspection and return the violated constraint $x_i \geq 0$. So, we can assume $\bar{x} \geq 0$. In this case, the oracle must only check for violated subtour elimination constraints (1.5.ii), that is it must solve the following problem:

Problem 1. *Find a set $\bar{S} \subseteq E$ such that $\bar{x}(E[\bar{S}]) > |\bar{S}| - 1$ or prove that $\bar{x}(E[S]) \leq |S| - 1$ for all $S \subseteq E$.*

We solve the above problem by linear programming. In particular, we first build a suitable 0-1 linear program and then show that we can drop the integrality constraint on the variables.

To this end, let $z \in \{0,1\}^V$ be the incidence vector of a set $S \subseteq V$. Observe that $w \in \{0,1\}^E$ is the incidence vector of $E[S]$ if and only if $w_{ij} = 1$ precisely when $z_i = 1$ and $z_j = 1$, whereas $w_{ij} = 0$ otherwise. This can be expressed by the following system of linear inequalities (plus the 0-1 constraints on the variables):

$$\begin{array}{ll} & w_e \leq z_i, \qquad w_e \leq z_j \quad \text{for } e = (i,j) \in E \\ SUB(V,E) & w_e \geq z_i + z_j - 1 \quad \text{for } e = (i,j) \in E \\ & w \in \{0,1\}^E, z \in \{0,1\}^V \end{array} \tag{1.12}$$

Denote by $SUB(V,E)$ the set of feasible solutions to (1.12). So $(\bar{z}, \bar{w})$ identifies a subtour elimination constraint associated to $\bar{S}$ and is violated by $\bar{x}$ if and only if

- $\bar{z}$ is the incidence vector of $\bar{S}$
- $(\bar{z}, \bar{w}) \in SUB(V,E)$ and
- $\sum_{e \in E[\bar{S}]} \bar{x}_e > |\bar{S}| - 1$.

By observing that $\sum_{e \in E[\bar{S}]} \bar{x}_e = \sum_{e \in E} \bar{x}_e \bar{w}_e$ and $|\bar{S}| = \sum_{i \in V} \bar{z}_i$, the last condition can be rewritten as follows:

$$\sum_{e \in E} \bar{x}_e \bar{w}_e > \sum_{i \in V} \bar{z}_i - 1 \tag{1.13}$$

We are finally able to write a separation oracle for the subtour elimination constraints. Define the following 0-1 linear optimization problem:

$$\begin{array}{l} \max \sum_{e \in E} \bar{x}_e w_e - \sum_{i \in V} z_i \\ s.t. \\ \quad (z,v) \in SUB(V,E) \end{array} \tag{1.14}$$

Denote by (z^*, w^*) an optimal solution to (1.14), and let v^* be its value. Then it is easy to see that

- if $v(z^*, w^*) > -1$, the (z^*, w^*) identifies a subtour elimination constraint violated by $\bar{x}$ which is returned by the oracle.
- if $v(z^*, w^*) \leq -1$ then no solutions in $SUB(V,E)$ satisfy (1.13) and the oracle returns no violated constraint.

The separation oracle just described requires the solution of the 0-1 linear program (1.14), which in turn may be difficult. However, the next theorem states that we can simply solve the *linear relaxation* of (1.14), that is, the linear program obtained from (1.14) by replacing the binary stipulation on the variables with the following linear (*box*) constraints:

$$0 \leq w_e \leq 1 \text{ for } e \in E, \quad 0 \leq z_i \leq 1 \text{ for } i \in V. \tag{1.15}$$

The proof of this theorem can be found in [23].

Theorem 1.5. *The linear relaxation of (*1.14*) always has an integer optimal solution solving (*1.14*).*

The dynamic simplex method

dynamic simplex method

Separation oracles and the simplex method are the building blocks of a dynamic simplex algorithm for the LP program (Q) defined as $\max\{w^T x : x \in P\}$, where $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is a polyhedron, with $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. In particular the dynamic simplex method solves a sequence of LP programs ($\mathrm{Q^i}$) defined as $\max\{w^T x : x \in P^i\}$, for $i = 1, \ldots, t$, with

$$P^1 \supset P^2 \supset \cdots \supset P^t \supseteq P.$$

This implies that for $i = 1, \ldots, t$, ($\mathrm{Q^i}$) is a relaxation of ($\mathrm{Q^{i+1}}$), and we have $v(\mathrm{Q}^i) \geq v(\mathrm{Q^{i+1}}) \geq v(\mathrm{Q})$. For $i = 1, \ldots, t$, we define $P^i = \{x \in \mathbb{R}^n : D^i x \leq d^i\}$, where $[D^i \; d^i]$ is a matrix obtained by considering a suitable subset of rows of $[A \; b]$. Also, $[D^{i+1} \; d^{i+1}]$ is obtained from $[D^i \; d^i]$ by including an additional row from $[A \; b]$. To simplify the discussion, we suppose that $P^1, \ldots, P^t, P$ are bounded, which implies that the associated linear programs are either empty or have an optimal solution.

The dynamic simplex method starts by selecting a suitable subset of rows $[A \; b]$, denoted by $[D^1 \; d^1]$. Then, at each iteration i, the linear program $\max\{w^T x : x \in P^i\}$ denoted by ($\mathrm{Q^i}$) is solved by the simplex method. If P^i is empty, then $P \subseteq P^i$ is also empty and the dynamic simplex method terminates. Otherwise, let $\bar{x}^i$ be an optimal solution to $(\mathrm{Q})^i$.

At this stage, we invoke a separation oracle to establish whether $\bar{x}^i \in P$ or not. If $\bar{x}^i \in P$, then $w^T \bar{x}^i \leq v(\mathrm{Q})$. Also, since ($\mathrm{Q}^i$) is a relaxation of ($Q$), we have $w^T \bar{x}^i = v(\mathrm{Q}^i) \geq v(\mathrm{Q})$: so, we can conclude that $w^T \bar{x}^i = v(\mathrm{Q})$ and $\bar{x}^i$ is optimal to (Q).

Finally, suppose $\bar{x}^i \notin P$. Then the separation oracle returns a constraint of the description of P violated by $\bar{x}^i$. We obtain $[D^{i+1}\ d^{i+1}]$ by adding this new row to $[D^i\ d^i]$ and iterate.

The idea behind the dynamic simplex method is that the algorithm terminates long before we have added all the defining constraints of P. Next, we show an application of the dynamic simplex method to solve the Maximum Weight Forest. This is only to illustrate the methodology with our example, as there exists a simpler and purely combinatorial algorithm to solve such problem.

Example 1.4. *Let P be the forest polytope associated to the graph in Figure* 1.

Let P^1 be obtained from P by dropping all subtour elimination constraints but those associated with the sets of two elements.

Then Problem (Q^1) is:

$$\begin{aligned}
&\max 4x_{12} - x_{13} + 3x_{14} + 4x_{24} + 3x_{25} - x_{34} + x_{36} + 2x_{45} - x_{46} - 2x_{56}\\
&s.t.\\
&\quad 0 \leq x_e \leq 1 \text{ for all } e \in E
\end{aligned}$$

The optimal solution is simply $x^1_{12} = x^1_{14} = x^1_{24} = x^1_{25} = x^1_{36} = x^1_{45} = 1$, and all other variables are 0, whose value is 17. The separation oracle applied to x^1 return $x^1 \notin P$ plus the violated subtour inequality $x_{12} + x_{14} + x_{24} \leq 2$, associated to $S = \{1, 2, 4\}$. This is added to the linear description of P^1 to obtain P^2 and the simplex algorithm resolves the following problem (Q^2):

$$\begin{aligned}
&\max 4x_{12} - x_{13} + 3x_{14} + 4x_{24} + 3x_{25} - x_{34} + x_{36} + 2x_{45} - x_{46} - 2x_{56}\\
&s.t.\\
&\quad x_{12} + x_{14} + x_{24} \leq 2\\
&\quad 0 \leq x_e \leq 1 \text{ for all } e \in E
\end{aligned}$$

An optimal solution to (Q^2) is $x^2_{12} = x^2_{24} = x^2_{25} = x^2_{36} = x^2_{45} = 1$ and all other variables are 0, with value 14.

The next invocation to the oracle produces the violated inequality $x_{24} + x_{25} + x_{45} \leq 2$, associated to $S = \{2, 4, 5\}$. We include it in P^3 and solve Q^3 obtaining $x^3_{12} = x^3_{14} = x^3_{25} = x^3_{36} = x^3_{45} = 1$, with value 13.

Next we add the violated inequality $x_{12} + x_{14} + x_{24} + x_{25} + x_{45} \leq 3$, associated to $S = \{1, 2, 4, 5\}$. We include it in P^4 and solve Q^4 obtaining $x^4_{12} = x^4_{14} = x^4_{25} = x^4_{36} = 1$, with value 12. We have $x^4 \in P$ and the algorithm terminates.

Note that it was sufficient to include only three subtour inequalities to find an optimal solution to Q, where P contains 32 non trivial such inequalities.

Chapter 2

Integral polyhedra, TDI systems and totally unimodular matrices

2.1 Integral polyhedra and TDI systems

Let $P \subseteq \mathbb{R}^n$ be a nonempty polyhedron. We define its *integer hull* P_I by

$$P_I = \text{conv}(P \cap Z^n) \tag{2.1}$$

so this is the convex hull of the intersection between P and the lattice Z^n of integral points. Note that P_I may be empty although P is not. If P is bounded, it contains a finite number of integral points, and therefore P_I is a polytope. By the finite basis theorem for polytopes (see Dahl [6]) it follows that P_I is a polyhedron. The next result (see [20]), says that the same conclusion holds even in the unbounded case, provided that P is a rational polyhedron (that is, defined by linear inequalities with rational data).

Theorem 2.1. *If P is a rational polyhedron, then P_I is a polyhedron. If P_I is nonempty,* $\text{char.cone}(P_I) = \text{char.cone}(P)$.

Proof. Let $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. According to the decomposition theorem for polyhedra we have that $P = Q + C$ where Q is a polytope and $C = \text{char.cone}(P) = \{x \in \mathbb{R}^n : Ax \leq 0\}$. Choose a finite set of *integral* generators $G = \{g_1, \ldots, g_m\}$ for C so $C = \text{cone}(G)$, and consider the set

$$M = \{\sum_{j=1}^{m} \mu_j g_j : 0 \leq \mu_j \leq 1 \text{ for } j = 1, \ldots, m\}. \tag{2.2}$$

M is a polytope, it coincides with the convex hull of the vectors $\sum_{j=1}^{m} \mu_j g_j$ where $\mu_j \in \{0, 1\}$ for $j = 1, \ldots, m$.

We shall prove that

$$P_I = (Q + M)_I + C. \tag{2.3}$$

Let p be an integral vector in P, so $p = q + c$ for some $q \in Q$ and $c \in C$. Then $c = \sum_{j \le m} \mu_j g_j = c' + c''$ where we define $c' = \sum_{j \le m} (\mu_j - \lfloor \mu_j \rfloor) g_j$ and $c'' = \sum_{j \le m} \lfloor \mu_j \rfloor g_j \in C$. Then c'' is integral and $c' \in M$ (as $0 \le \mu_j - \lfloor \mu_j \rfloor \le 1$). This gives $p = (q+c')+c'' \in (Q+M)_I + C$ because $q+c' \in Q+M$ and $q+c' = p-c''$ which is integral. We have therefore shown that $P_I \subseteq (Q+M)_I + C$. Furthermore, since $Q+M \subseteq P$ and $C = C_I$ we obtain $(Q+M)_I + C \subseteq P_I + C_I \subseteq (P+C)_I = P_I$. This proves (2.3).

The proof can now be completed by applying the decomposition theorem for polyhedra. In (2.3) $Q+M$ is a bounded polyhedron (i.e., a polytope) and therefore $(Q+M)_I$ is polytope. Thus, by (2.3), P_I is algebraic sum of a polytope and a convex cone which proves that it is a polyhedron. Furthermore, if P_I is nonempty, we also get char.cone(P_I) = char.cone(P) (from the uniqueness of polyhedral decomposition).

□

A polyhedron is called *integral* if $P = P_I$, i.e. it equals the convex hull of its integral vectors. (For convenience, we also say that the empty polyhedron is integral.) Integral polyhedra are interesting in connection with integer linear programming. In fact, we have in general that

$$\begin{aligned} &\max\{c^T x : x \in P, x \text{ is integral}\} = \\ &\max\{c^T x : x \in P_I\} \le \\ &\max\{c^T x : x \in P\}. \end{aligned} \tag{2.4}$$

If P is integral the inequality in (2.4) can be replaced by equality, and the values of the ILP and the corresponding LP coincide. In fact, among the optimal solutions of the LP problem $\max\{c^T x : x \in P\}$ there is an integral one.

Some equivalent descriptions of integral polyhedra are listed next; the proof is left for an exercise.

Proposition 2.2. *The following conditions are all equivalent whenever $P \subseteq \mathbb{R}^n$ is a nonempty polyhedron.*

(i) P is integral.

(ii) Each minimal face of P contains an integral vector.

(iii) $\max\{c^T x : x \in P\}$ is attained for an integral vector for each $c \in \mathbb{R}^n$ for which the maximum is finite.

Furthermore, if P is pointed (e.g., if $P \subseteq \mathbb{R}^n_+$), then P is integral if and only if each vertex is integral.

There are some further equivalent conditions describing the integrality of a polyhedron. One such interesting condition is that the optimal *value* is integral for any LP over the polyhedron with integral objective function. This result leads to the concept of total dual integrality which gives a method for proving that a polyhedron is integral.

Proposition 2.3. *Let P be a rational polyhedron in $\mathbb{R}^n$. Then P is integral if and only if $\max\{c^T x : x \in P\}$ is an integer for each $c \in Z^n$ such that $v(LP)$ is finite.*

Proof. In order to simplify a bit, we only prove this result for the situation with P pointed. Assume that P is integral, and consider $\max\{c^T x : x \in P\}$ with $c \in Z^n$ and assume that the optimal value is finite. Then, from Proposition 2.2 this LP has an optimal solution x^* which is integral. But then clearly the optimal value $c^T x^*$ is an integer, which proves the "only if" part.

We prove the converse implication by contradiction. So assume that P is not integral. As P is pointed, and rational, this means that there is a vertex $\bar{x}$ of P with a fractional component $\bar{x}_j$. (Add details .....). Therefore, there is a $\bar{c} \in Z^n$ such that $\bar{x}$ is the *unique* optimal solution of $\max\{\bar{c}^T x : x \in P\}$. One can then show that there is an $\epsilon > 0$ such that for each $c' \in B(\bar{c}, \epsilon)$ $\bar{x}$ is the unique optimal solution of the problem $\max\{(c')^T x : x \in P\}$. For a suitably large positive integer s we have that $d = \bar{c} + (1/s)e_j \in B(\bar{c}, \epsilon)$. Note that $\bar{x}$ is the optimal solution of the LP problem over P with objective function $sd = s\bar{c} + e_j$ and therefore its optimal value equals $sd^T\bar{x} = s\bar{c}^T\bar{x} + \bar{x}_j$. But then one of the two optimal values $sd^T\bar{x}$ and $s\bar{c}^T\bar{x}$ must be fractional (as $\bar{x}$ is fractional), i.e., there is an integer objective function such that the corresponding optimal value is fractional; a contradiction. This completes the proof.

□

We call a linear system $Ax \leq b$ *totally dual integral*, or *TDI* for short, if for all integral c with $\max\{c^T x : Ax \leq b\}$ finite, the dual LP problem $\min\{y^T b : y^T A = c^T, y \geq 0\}$ has an integral optimal solution. Note here that this definition concerns a given *linear system* and not the corresponding polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. In fact, a polyhedron may be represented by both a TDI system and another non-TDI system. The importance of the TDI property is seen from the next result.

Corollary 2.4. *Assume that the system $Ax \leq b$ is TDI and that b is integral. Then the associated polyhedron $P = \{x : Ax \leq b\}$ is integral.*

Proof. By definition of TDI the dual problem (D) $\min\{y^T b : y^T A = c^T, y \geq 0\}$ has an integer optimal solution y^* for each $c \in Z^n$ with $\max\{c^T x : Ax \leq b\}$ finite.

But as b is integral we get that the optimal value $v(D) = (y^*)^T b$ is also integral for such problems. By the LP duality theorem, this shows that $\max\{c^T x : x \in P\}$ is an integer for each $c \in Z^n$ such that $v(LP)$ is finite, and P is integral by Proposition 2.3.

□

Note here that the fact that $Ax \leq b$ is TDI is not sufficent to guarantee that $P = \{x : Ax \leq b\}$ is integral, because with fractional b the optimal value of the dual problem may be fractional (although the optimal solution is integral). It can be shown that if $Ax \leq b$ is a rational linear system, then there is an integer M such that the scaled system $(1/M)Ax \leq (1/M)b$ is TDI. As a consequence, we see that each rational polyhedron has a TDI representation.

An example of a TDI system is

$$\begin{array}{c} x(\delta^+(U)) \geq 1 \quad \text{for all } U \subset V,\, r \in U; \\ x \geq 0 \end{array} \tag{2.5}$$

where r is a given node in a directed graph $D = (V, E)$. The solution set of this system is the dominant of the convex hull of incidence vectors of r-arborescences in D. An *r-arborescence* is an arc set F such that the subgraph (V, F) contains a directed rt-path for each node $t \neq r$ and where each such node t has exactly one ingoing arc in F. We remark that these graphs are of interest in the design of certain transportation or communication networks.

2.2 Totally unimodular matrices

In the previous section we studied the integrality of a polyhedron in connection with TDI linear systems. The topic of the present section is to study a stronger property, namely for the matrix A alone, which assures the integrality of the associated polyhedron.

An $m \times n$ matrix A is called *totally unimodular*, *TU* for short, if the determinant of each square submatrix is -1, 0 eller 1. Thus, in particular, a TU matrix has entries being -1, 0 or 1. Such matrices arise naturally in connection with graphs, see Section 2.3.

A relation between integrality of a polyhedron and total unimodularity is given next.

Theorem 2.5. *Let $A \in \mathbb{R}^{m,n}$ be a TU matrix and $b \in \mathbb{R}^m$ an integral vector. Then the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ is integral.*

Proof. First we recall Cramer's rule for finding the inverse of a nonsingular matrix C: the (j,i)'th element of C^{-1} is given by $(-1)^{i+j}\det(C_{ij})/\det(C)$ where C_{ij} is obtained from C by deleting the i'th row and the j'th column. It follows that if C is integral and $\det(C)$ is either 1 or -1, then C^{-1} will be integral.

Let $F = \{x \in \mathbb{R}^n : A'x = b'\}$ be a minimal face of P (so $A'x \leq b'$ is a subsystem of $Ax \leq b$). We may assume that the m' equations in $A'x \leq b'$ (or the rows of A') are linearly independent (otherwise we could remove redundant constraints without changing the solution set). Then A' contains at least one $m' \times m'$ nonsingular submatrix B so (after permutation of columns) we may write $A' = \begin{bmatrix} B & N \end{bmatrix}$. Therefore a vector x in F is given by $x_B = B^{-1}b'$, $x_N = 0$ where x_B and x_N are the subvectors corresponding to B and N, respectively. But since B is a nonsingular submatrix of A and A is TU, it follows that B must be integral and its determinant is 1 or -1. Then, by Cramer's rule, B^{-1} is integral, and therefore x_B is integral. Thus F contains an integral vector x which proves that P is integral.

□

The TU property is preserved under a number of matrix operations, for instance

- transpose;
- augmenting with the identity matrix;
- deleting a row or column which is a coordinate vector;
- multiplying a row or column by -1;
- interchanging two rows;
- duplication of rows or columns.

We leave the proof of these facts for an exercise.

An important connection between integrality for dual LP problems and total unimodularity is discussed next.

Corollary 2.6. *Let $A \in \mathbb{R}^{m,n}$ be a TU matrix and $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ two integral vectors. Then each of the dual LP problems in the duality relation*

$$\max\{c^T x : Ax \leq b\} = \min\{y^T b : y^T A = c^T,\ y \geq 0\}. \tag{2.6}$$

has an integral optimal solution.

Proof. From Theorem 2.5 it follows that the primal polyhedron $\{x \in \mathbb{R}^n : Ax \leq b\}$ is integral. Therefore, by Proposition 2.2, the primal LP has an integral optimal solution. The dual LP problem may be viewed as the problem $\min\{b^T y : Dy \leq d\}$ where D is given by $D = \begin{bmatrix} A^T \\ -A^T \\ -I \end{bmatrix}$. and $d = \begin{bmatrix} c \\ -c \\ 0 \end{bmatrix}$. Note that D is obtained

from A by using operations that preserve the TU property (see above), so D is TU. Since d is integral, the dual polyhedron is integral and the dual LP has an integral optimal solution.

□

Corollary 2.7. *Assume that $A \in \mathbb{R}^{m,n}$ is a TU matrix. Let b, b', d, d' be integral vectors with $b \leq b'$ and $d \leq d'$ where we allow components to be either $-\infty$ or ∞. Then the polyhedron $P = \{x \in \mathbb{R}^n : b' \leq Ax \leq b,\ d' \leq x \leq d\}$ is integral.*

Proof. We have that $P = \{x : Cx \leq c\}$ where

$$C = \begin{bmatrix} A \\ -A \\ I \\ -I \end{bmatrix}, \quad c = \begin{bmatrix} b \\ -b' \\ d \\ -d' \end{bmatrix}.$$

Note that whenever a component of b, b', d, d' is $-\infty$ or ∞, the corresponding constraint is dropped. Now, C is TU as it is obtained from A by TU preserving operations and also c is integral, so P is integral according to Theorem 2.5.

□

As we have seen, polyhedra defined by a TU matrix are integral. An interesting converse result due to Hoffman and Kruskal (1956) is given next without proof.

Theorem 2.8. *Suppose that A is an integral matrix. Then A is TU if and only if the polyhedron $\{x : Ax \leq b,\ x \geq 0\}$ is integral for every integral b.*

In order to determine if a matrix is TU the following criterion due to Ghouila-Houri (1962) may be useful. For a proof, see e.g. [15].

Proposition 2.9. *A $(0, \pm 1)$-matrix (of size $m \times n$) A is TU if and only if for each $J \subseteq \{1, \ldots, n\}$ there is a partition J_1, J_2 of J (where J_1 or J_2 may be empty) such that*

$$|\sum_{j \in J_1} a_{ij} - \sum_{j \in J_2} a_{ij}| \leq 1 \quad \text{for } i = 1, \ldots, m. \tag{2.7}$$

2.3 Applications: network matrices and minmax theorems

We give some basic examples of TU matrices in connection with graphs, and derive important combinatorial minmax theorems.

Consider first an (undirected) graph $G = (V, E)$ and let $A_G \in \{0, 1\}^{V \times E}$ be its node-edge incidence matrix, i.e., the column of A_G corresponding to the edge $e = [u, v]$ has only two nonzeros, namely for the two rows corresponding to u and

v. Note that since A_G is an incidence matrix (all elements are 0 or 1), it *may* be totally unimodular. The precise answer is that it is TU exactly when the graph contains no odd cycles.

Proposition 2.10. *A_G is TU if and only if G is bipartite.*

Proof. Let first G be bipartite with the two color classes I_1 and I_2. We shall then show that A_G is TU using Ghouila-Houri's TU characterization. Let I be a subset of the rows of A_G, and let $I_1' = I \cap I_1$ and $I_2' = I \cap I_2$. Let a be the vector obtained by summing the rows of A_G associated with I_1' and subtracting the rows associated with I_2'. The component a_e of a correponding to an edge $e = [u, v]$ must be either 1 (if one of the endnodes is in I_1' and the other is not in I_2'), -1 (the converse situation) or 0 (if both or none of the endnodes lie in I). This a is a vector with componenents -1,0 and 1 and therefore A_G is TU according to Proposition 2.9.

Conversely, assume that G is not bipartite. It follows that G has an odd cycle C. Let B be the square submatrix of A_G indiced by the rows and columns corresponding to the nodes and edges of C, respectively. With suitable permutations we transform this matrix into the circulant matrix $C_{2,t}$ where t is the length of the cycle. (This is a 0,1-matrix where the i'th row, for $i = 1, \ldots, t-1$ has two nonzeros (1's) namely in position i and $i+1$, while the two nonzeros (1's) of row t are in the first and last position). One can show that $|\det(C_{2,t})| = 2$ and it follows that A_G is not TU.

□

The previous result may be combined with Corollary 2.6 to get important combinatorial minmax theorems. We illustrate this for problems concerning packing and/or covering of nodes and edges of a graph. First, we give a more "symmetric" integrality/LP theorem derived from Corollary 2.6.

Corollary 2.11. *Let $A \in \mathbb{R}^{m,n}$ be a TU matrix and $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ two integral vectors. Then each of the dual LP problems in the duality relation*

$$\max\{c^T x : Ax \leq b,\ x \geq 0\} = \min\{y^T b : y^T A \geq c^T,\ y \geq 0\}. \tag{2.8}$$

has an integral optimal solution provided that the optimal values are finite.

Proof. We apply Corollary 2.6 to the matrix $\begin{bmatrix} A & -I \end{bmatrix}$ which is TU as A is TU. The primal problem becomes $\max\{c^T x : Ax \leq b,\ x \geq 0\}$ as desired. The dual problem is

$$\begin{aligned} &\min\{y^T b : y^T A - z = c^T,\ y \geq 0,\ z \geq 0\} = \\ &\min\{y^T b : y^T A \geq c^T,\ y \geq 0\}. \end{aligned}$$

Note here that there is an optimal integral solution (y, z) of the first problem if and only if there is an optimal integral solution y of the second problem. The result then follows from the duality relation.

□

Let G be a bipartite graph, so by the previous proposition, its incidence matrix A_G is TU. Consider the LP duality relation (2.8) with A replaced by A_G and with $c = \mathbf{1}$ and $b = \mathbf{1}$. We then obtain

$$\max\{\mathbf{1}^T x : A_G x \leq \mathbf{1},\ x \geq 0\} = \min\{y^T \mathbf{1} : y^T A_G \geq \mathbf{1}^T,\ y \geq 0\}. \tag{2.9}$$

We know that each of these two problems has an integral optimal solution and we interprete this combinatorially. First, we note that in the system $Ax \leq \mathbf{1},\ x \geq 0$ we have one variable x_e for each edge $e \in E$, and that x is a solution if and only if x is nonnegative and satisfies $x(\delta(v)) \leq 1$ for each $v \in V$. Thus an integer solution here must in fact be 0,1-valued, and it represents a matching in G. A *matching* in a graph is an edge subset such that no node is incident to more than one edge. Therefore, the maximization problem in (2.9) is to find a maximum cardinality matching in the bipratite graph G. This is a classical problem which is polynomially solvable by e.g. a combinatorial algorithm called the Hungarian algorithm. Now, we turn to the minimization problem, and observe that there is only one variable x_v for each node $v \in V$. An *integer* feasible solution of this problem assigns a nonnegative integer to each node in such way that each edge has an endnode with a strictly positive integer. Clearly, we may restrict our attention to such integers that are 0 or 1, and the the minimization problem becomes: find a *node cover* in G of minimum cardinality. A node cover is a subset V_0 of the nodes such that each edge has at least one endnode in V_0. Therefore, due to integrality, the relation (2.9) says that *the maximum cardinality of a matching in a bipartite graph equals the minimum cardinality of a node cover*. This result is known as the *König-Egervary theorem* and dates to 1931.

As a second application of Corollary 2.11 we study the effect of the choice $A = A_G^T$, i.e., the transpose of the node-edge incidence matrix of G. Again we use $c = \mathbf{1}$ and $b = \mathbf{1}$. We shall assume that G contains no isolated node (otherwise one of the problems studied would become infeasible). We then obtain (when we let the role of x and y be changed)

$$\max\{\mathbf{1}^T y : A_G^T y \leq \mathbf{1},\ y \geq 0\} = \min\{x^T \mathbf{1} : x^T A_G^T \geq \mathbf{1}^T,\ x \geq 0\}. \tag{2.10}$$

We recall the interpretations above and see that an integral feasible y in the maximization problem corresponds to a *node packing* (also called *independent set* or *stable set*). Thus this problem may be viewed as to find a maximum cardinality

node packing in G. A node packing is a subset S of the node set such that no pair of nodes are adjacent. As remarked above in connection with the node cover problem, one may restrict the attention to 0-1 valued solutions in the minimization problem of (2.10). The problem therefore reduces to that of finding a minimum cardinality edge cover. An *edge cover* is a subset of the edge set such that each node is incident to one of the chosen edges. The interpretation of the minmax relation of (2.10) then becomes: *the maximum cardinality of a node packing in a bipartite graph equals the minimum cardinality of an edge cover*. This result is also due to König (1933) and is often called *König's covering theorem*.

Note that all of the four combinatorial problems discussed above are polynomially solvable in bipartite graphs. In fact, it follows from the results above that they may be solved by a polynomial algorithm for LP which finds optimal vertex solutions (and such algorithms do exist, e.g., based on the ellipsoid method). There are also purely combinatorial algorithms for each of these problems that are polynomial, see e.g. [14].

We next study problems in directed graphs. Let $D = (V, E)$ be a directed graph and let A_D be its node-arc incidence matrix.

The basic tool is the following result.

Proposition 2.12. *A_D is TU for each digraph D.*

Proof. We give an elegant and short proof of this fact due to Veblen and Franklin (1921), see [20].

We prove by induction that each subdeterminant is -1, 0 or 1. Assume that this is true for all submatrices of dimension t, and let N be a $t \times t$ submatrix of A_D. If N contains a column with all zeros, then $\det(N) = 0$. If a column of N contains exactly one nonzero, this number is either -1 or 1, and when we expand the determinant for this column we get that $\det(N) \in \{-1, 0, 1\}$ by the induction hypothesis. Finally, if each column of N contains two nonzeros, the sum of all row vectors is the zero vector, so the row vectors ar linearly dependent and $\det(N) = 0$.

□

Using this fact combined with Corollary 2.11 we obtain that

$$\begin{array}{l} \max\{c^T x : A_D x = 0,\ 0 \le x \le d,\ \ x \text{ integral}\} = \\ \min\{y^T d : z^T A_D + y \ge c^T,\ y \ge 0,\ y, z \text{ integral}\}. \end{array} \tag{2.11}$$

We here assume that c and d are integral vectors and allow components of d to be ∞ (in which case the corresponding dual variable is not present). An interpretation may be given as follows. Recall that $x \in \mathbb{R}^E$ is called a circulation if

$A_D x = 0$. Thus the primal problem is to find a nonnegative integral circulation in D obeying arc capacities d and maximizing the "profit" $\sum_{e \in E} c_e x_e$. In the dual problem we may interpret z as node variables z_v for $v \in V$ and then the constraints $z^T A_D + y \geq c^T$ says that $z_u - z_v + y_e \geq c_e$ for each arc $e = uv \in E$.

Consider next the special case when the profit function is all zero except for on one arc (t, s) where $c_{ts} = 1$. In addition we let $d_{ts} = \infty$. Then the circulation problem coincides with the maximum st-flow problem. Thus we see (due to Corollary 2.11) that the maximum flow problem has an integral optimal solution. Furthermore, (y, z) is feasible in the dual problem if and only if (y, z) is integral, y nonnegative and

$$\begin{array}{ll} z_u - z_v + y_e \geq 0 & \text{for all } e = (u, v) \neq (t, s); \\ z_t - z_s \geq 1 & \text{for all } e \neq (t, s) c_e. \end{array} \tag{2.12}$$

Note that the last inequality is due to the fact that $d_{ts} = \infty$. Therefore $z_t \geq z_s + 1$. Define the node set $W = \{v \in V : z_v \geq z_t\}$ and note that $t \in W$ while $s \notin W$. In addition we have for each $e = (u, v) \in \delta^-(W)$ that $z_v \geq z_t$, $z_u \leq z_t - 1$ (due to integrality) so from feasibility we get $y_e \geq z_v - z_u \geq z_t - z_t + 1 = 1$. Since y is nonnegative we therefore obtain $y^T d \geq \sum_{e \in \delta^-(W)} y_e d_e \geq \sum_{e \in \delta^-(W)} d_e$. This proves that the optimal value of the dual problem is no less than the capacity $d(\delta^-(W))$ of the cut $\delta^-(W)$. On the other hand the value of the primal problem is not larger than $d(\delta^-(W)$. But from (2.11) the values of these two problems coincide, and it follows that there exists an st-cut with capacity equal to the maximum flow from s to t. Thus we have given an alternative proof of the max-flow min-cut theorem.

Chapter 3

Heuristic algorithms for combinatorial optimization problems

In this chapter we describe a class of solution methods for combinatorial optimization problems, i.e. problems of type:

$$\max\{w(F) : F \in \mathcal{F}\} \tag{3.1}$$

where $F \subseteq E$ (ground set) and $w : E \to \mathbb{R}$.

In Chapter 1 we discussed the relevance of attacking the combinatorial optimization problem from two sides, by computing both upper and lower bounds on the optimal value to (3.1). In this chapter we focus on methods to compute lower bounds, so called *heuristics*. Heuristic methods are typically designed to find good, but not necessarily optimal, solutions quickly. Depending on the input, heuristics maybe classified in *constructive heuristics*, which generate a solution from scratch, and *improvement heuristics*, which start from a given feasible solution and try to improve it, typically by applying some type of local changes which modify only parts of the original solution.

3.1 Constructive heuristics

Constructive heuristics are designed to produce an initial solutions to an optimization problem. This may be difficult, depending on the problem characteristics. Many problems, however, have a certain monotonicity property that helps. We say that $\mathcal{F}$ is an *independence system* if each subset of a feasible set is also a feasible set. For instance, the set of forests in a graph is an independence system. We should remark that any set system (class of subsets of E) can be made into an independence system by adding all subsets of feasible sets; this is called

monotonization. This transformation may, at least, be useful in the analysis of the problem. For instance, in the Travelling Salesman Problem, we may consider the class of edge sets being subsets of Hamilton cycles.

For an independence system the *greedy algorithm* may be used to construct a feasible solution:

Greedy algorithm

Step 1. Order the elements in E in a sequence $e_1, \ldots, e_m$ such that $w(e_1) \geq \ldots \geq w(e_m)$. Set $F := \emptyset$.

Step 2. While $F \in \mathcal{F}$ add the next element in the sequence to F.

It is easy to see that Kruskal algorithm for spanning trees ([21]) is a greedy algorithm.

This algorithm terminates with a feasible solution, and it is simply called the *greedy solution*.

Example 3.1. *Consider the following combinatorial optimization problem. Let the ground set be* $E = \{1, 2, 3, 4\}$*, and let* $w(1) = 4$*,* $w(2) = 3$*,* $w(3) = 2$*,* $w(4) = -8$*. Let* $\mathcal{F} = \{\{1, 3\}, \{1, 3, 4\}, \{1, 2, 4\}, \{2, 3\}\}$.

After monotonization, the extended solution set becomes $\mathcal{F}^* = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, , \{2, 3\}, \{2, 4\}, \{3, 4\}\ \{1, 3, 4\}, \{1, 2, 4\}\}$.

According to the weight function, the ordered set of elements is $\{1, 2, 3, 4\}$. If we apply the greedy algorithm, then at the first iteration we have $F = \{1\}$. Next, we have $F = \{1, 2\}$. Finally, we have $F = \{1, 2, 4\}$. No more elements can be included in F to get a feasible solution and we are done. Remark that $w(F) = -1$ and the optimal solution is $F^* = \{1, 3\}$, with $w(F^*) = 6$.

Example 3.2. *Consider the weighted graph in Figure* 3.1*. Let us apply the greedy algorithm to the TSP problem introduced in Chapter* 1*.*

The sequence produced by the greedy algorithm will be $\{\}, \{(3, 5)\}$*,* $\{(3, 5), (2, 4)\}$*,* $\{(3, 5), (2, 4), (4, 5)\}$*,* $\{(3, 5), (2, 4), (4, 5), (1, 2)\}$*, and, finally* $H = \{(3, 5), (2, 4), (4, 5), (1, 2), (1, 3)\}$.

We have $w(H) = 26$, which can be easily verified to be non-optimal.

So, in general the greedy algorithm does not terminate with an optimal solution. However, it can be shown (see [21]) that the greedy solution is optimal (for every possible weight function w) if and only if the independence system is also a so-called *matroid*.

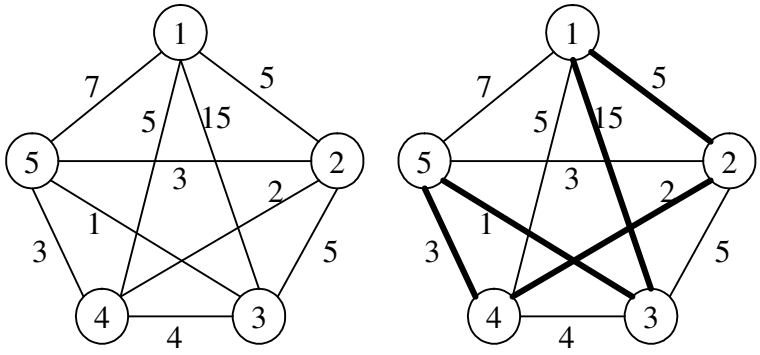

Figure 3.1: The greedy solution of Example 3.2

The greedy algorithm is probably the most simple and natural constructive heuristic for combinatorial optimization problems, but there are other approaches as well, often more effective.

approximation guarantee

We illustrate one such approach to the *metric* TSP problem, due to Christofides [3]. This method has an additional, interesting feature as it belongs to the class of heuristics with *approximation guarantee*. To introduce this concept, let (Q) be a maximization problem and let $v(\mathrm{Q})$ be its optimal value. Let $\mathcal{A}$ be an approximation algorithm for (Q), i.e. $\mathcal{A}$ finds a feasible solution x to (Q), and let $w(x)$ denote the value of x. Then $\mathcal{A}$ has a α-approximation factor guarantee if and only if $\alpha w(x) \geq v(\mathrm{Q})$. When (Q) is a minimization problem, the condition becomes $w(x) \leq \alpha v(\mathrm{Q})$.

Let $G = (V, E)$ be an undirected, complete graph and let $w : E \to \mathbb{R}$ be a weight function. We also suppose that w satisfies the triangle inequality, that is for all $u, v, z \in V$ we have $w(u, v) \leq w(u, z) + w(z, v)$.

In order to describe Christofides approximation algorithm for TSP, we need to introduce the definition of Eulerian tour in a (multi-graph).

Definition 3.1. *Let $G = (V, E)$ be a connected multigraph (edge repetitions are allowed). A* Eulerian *tour is a tour in G passing through each edge exactly once.*

One can show that a graph admits a Eulerian tour if and only if every node has even degree[1]. In this case, the graph is said to be *Eulerian*. A Eulerian tour can be easily constructed in a Eulerian graph.

1. Actually, this problem is considered as the *mother* of all network problems. It was first stated and solved by Euler while attacking the famous Königsberg 7-Bridges Problem

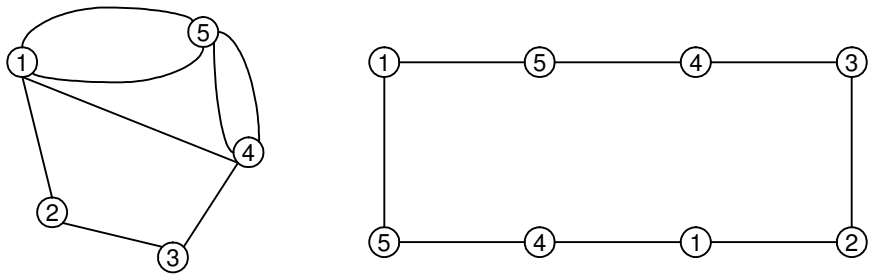

Figure 3.2: A Eulerian graph and a corresponding Eulerian tour

The following algorithm finds a factor 2-approx Hamilton tour H in G, i.e. if H^* is an optimal Hamilton tour, then $w(H) \leq 2w(H^*)$.

Christofides Algorithm

Step 1. Find a minimum spanning tree T of G.

Step 2. Double every edge of T to obtain a Eulerian graph

Step 3. Find a Eulerian tour $\mathcal{T}$ on this graph

Step 4. Return the Hamilton tour that visits the vertices of G in the order of their first appearance in $\mathcal{T}$.

Theorem 3.1. *Christofides algorithm is a factor 2 approximation algorithm for the metric TSP.*

Proof. Denote by H^* an optimal Hamilton tour in G and let $\bar{H}$ the tour returned by the algorithm. First observe that

(i) $w(T) \leq w(H^*)$. In fact, every Hamilton tour H contains a spanning tree: take any edge e of H and remove it from H; the remaining graph is a Hamilton path (a path passing through each node exactly once) and thus it is a spanning tree. (i) follows from the non-negativity of $w(e)$.

At Step 2, observe that $\mathcal{T}$ contains each edge of T exactly twice, hence $w(\mathcal{T}) = 2w(T)$.

Now, if $\mathcal{T}$ does not contain repeated nodes, then $\mathcal{T}$ is a Hamilton tour and $\bar{H} = \mathcal{T}$. It follows that $w(\bar{H}) = w(\mathcal{T}) = 2w(T) \leq 2w(H^*)$ (the last inequality holds from

(i), and the theorem holds.

If $\mathcal{T}$ contains repeated nodes, apply the following procedure to obtain $\bar{H}$

Shortcut repeated nodes:

Step 1. Identify a repeated node v in $\mathcal{T}$. Let u the previous node on $\mathcal{T}$ and let z be the next.

Setp 2. Replace in $\mathcal{T}$ the path $u, (u, v), v, (v, z)$ with the path $u, (u, z), z$.

Step 3. If $\mathcal{T}$ does not contain repeated nodes, let $\bar{H} = \mathcal{T}$ and terminate. Otherwise go to Step 1.

It is not difficult to see that, due to the triangle inequality, at each iteration of the above procedure the weight of $\mathcal{T}$ does not increase. It follows that $w(\bar{H}) \leq 2w(T) \leq 2w(H^*)$. □

An Example of application of Christofides algorithm is shown in Figure 3.3 and Figure 3.4.

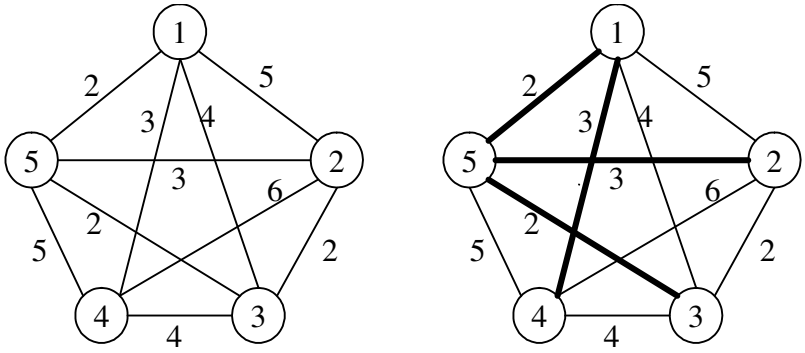

Figure 3.3: Step 1 of Christofides Algorithm: build a Minimum Spanning Tree

3.2 Improvement heuristics

An improvement heuristics finds a solution to a combinatorial optimization problem by iteratively trying to improve a given feasible solution. The most known, and popular, of these methods is the so called *Local Search*. It relies on the concept of neighborhood functions.

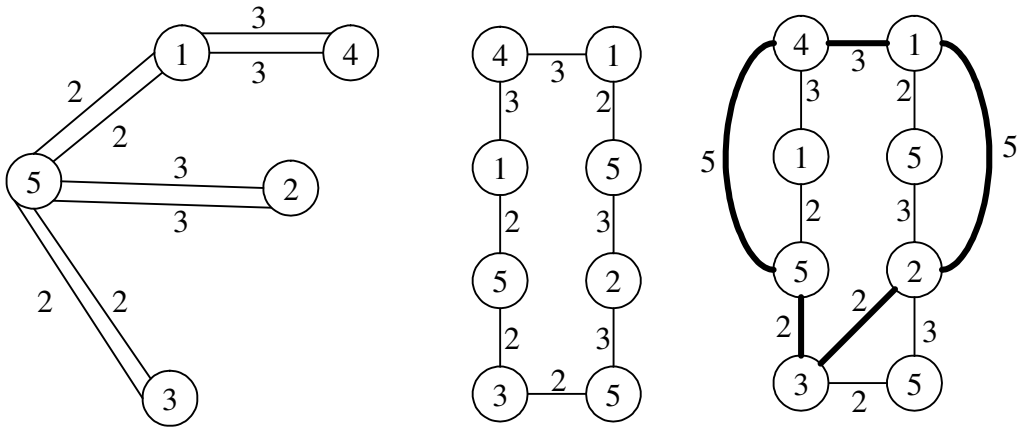

Figure 3.4: Step 2, 3 and 4 of Christofides Algorithm: the final Hamilton tour is shown in solid lines

Definition 3.2. *Let $\mathcal{F}$ be a class of subsets of a ground set E. A* neighborhood function *is a point to set map $N : \mathcal{F} \to 2^E$, which associates with each solution $F \in \mathcal{F}$ a subset of solutions $N(F) \subseteq \mathcal{F}$.*

neighborhoods

The set $N(F)$ is called *neighborhood* of F (under the function N). Neighborhood functions are used in a widely exploited technique to solve combinatorial optimization problems, known as *Local Search*. The idea is to start from an initial solution $\bar{F}$ and then search for a best solution F' in its neighborhood $N(\bar{F})$. If $w(F') \leq w(\bar{F})$, then F' is not improving over $\bar{F}$ and we stop. Otherwise, we take F' as a new starting solution and iterate.

Local Search

Step 0. Start with an initial solution $\bar{F}$.

Step 1. Let $F' = \text{argmax}\{w(F) : F \in N(\bar{F})\}$.

Step 2. If $w(F') > w(\bar{x})$ then let $\bar{F} := F'$ and goto Step 1. Else Stop

Note that when the local search terminates, $\bar{F}$ is an optimal solution to the restricted problem

$$\max\{w(x) : x \in N(\bar{x})\} \tag{3.2}$$

and we say that $\bar{F}$ is a *local optimal solution*. Of course, local optima may be very far from global optima. One may object that we have gone from solving a single combinatorial optimization problem to solving a whole sequence of combinatorial

optimization problems, whose solution does not even ensure global optimality. Is there any advantages in doing so? Of course, it all depends on our capacity to efficiently solve the optimization problem (3.2), and on the quality of the final solution. Concerning quality, this is typically measured by exploiting well established benchmark test-beds (there are relevant issues concerning this question, but we will not discuss them here). There are two major lines of attack to reach efficiency. The most common one is based on the definition of suitably "small" neighborhoods, so that the optimal solution can be found by complete enumeration: we call such approach *small neighborhood search*. The second approach is somehow smarter. Indeed, there are several combinatorial optimization problems which can be solved efficiently, even though the number of solutions is very large. Examples are the minimum spanning tree problem, the matching problem, the minimum path problem, etc. So, the game here is to define the neighborhood function N in such a way that the corresponding optimization problems (3.2) falls into this category. This is subject of the *exponential neighborhood search* theory.

Small Neighborhood Search. There is a huge (and perhaps boring?) literature on the definition of small but still effective neighborhoods for myriad versions (of versions) of combinatorial optimization problems. However, a few of these neighborhood definitions are quite interesting and very effective in practice, and provide the basis for various extensions. One such example is the 2-exchange neighborhood for the TSP introduced by Lin and Kerninghan [12]. We discuss this neighborhood in detail hereafter.

move

Typically, small neighborhoods are defined through moves. A *move* is an algorithm, which receives a feasible solution $S \in \mathcal{F}$ to a combinatorial optimization problem plus some additional input parameters, and returns a new solution T obtained from S by applying some small change to S. Moves are normally such that the *distance* $d(S,T)$ between S and T is small, where $d(S,T) = |S \Delta T|$. In other words, S and T are not *too* different from each other. The neighborhood of S is then defined as the set of solutions T which may be obtained by invoking the move on S with different additional input parameters. The neighborhood proposed by Lin and Kerninghan for the TSP belongs to the class of *distance-k-neighborhood*, which is defined for each $S \in \mathcal{F}$ as $N_k(S) = \{T \in \mathcal{F} : d(S,T) \leq k\}$. In particular, they implemented a distance-4-neighborhood which corresponds to applying a particular move called $2-exchange$ (the derived local search algorithm is called $2-opt$). A 2-exchange move is showed in Figure 3.5.

2-exchange heuristic

The 2-exchange move consists in selecting two non-adjacent edges from the given Hamilton tour S and replace them with two new edges so as to obtain a distinct

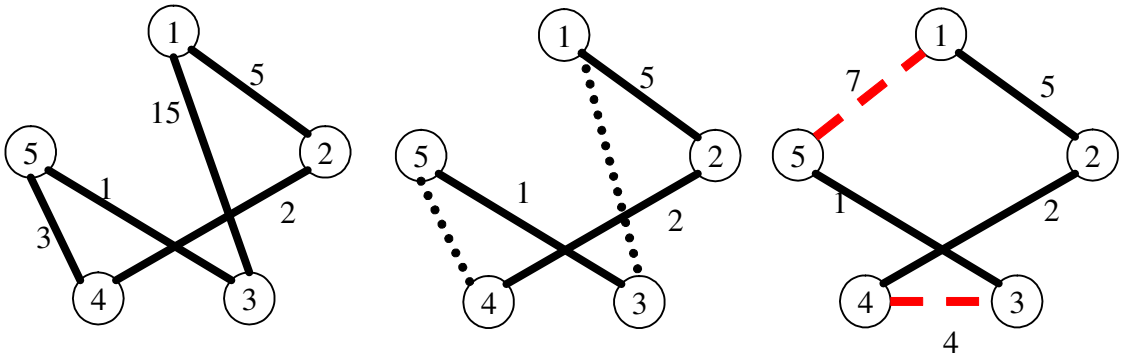

Figure 3.5: A 2-exchange move

Hamilton tour T. It is easy to see that, once we have chosen the two exiting edges, the entering pair is fixed. In the example above, if we remove edge $(4, 5)$ and edge $(1, 3)$ then the entering edges are $(1, 5)$ and $(3, 4)$ (remark that the initial tour is valued 26, whereas the final one is valued 18, which means that the move is improving). The final neighborhood is obtained by choosing every possible pair of non-adjacent edges and applying to each pair the 2-exchange move. Observe now that any Hamiltonian tour S contains $|V|$ edges. There are roughly $O(|V|^2)$ ways to select a pair of non-adjacent edges, which in turn implies that $|N(S)|$ is also $O(|V|^2)$, for each Hamilton tour S. So, even when V is large, such a neighborhood can be easily constructed explicitly and the optimal solution can be found by enumeration.

Exponential Neighborhood Search. Clearly, the larger the neighborhood, the more likely we generate good quality solutions. As an extreme case, if $N(S) = \mathcal{F}$, then the restricted optimization problem (3.2) actually coincides with the original problem (3.1). However, when we deal with large neighborhoods, complete enumeration becomes impractical and we need to resort to smarter search techniques. More specifically, a careful definition of the neighborhood function can allow us to resort to efficient solution algorithms, even if the number of solutions in the neighborhood grows exponentially with the input size. We discuss an example for the TSP introduced by Sarvanov and Doroshko [19].

Let $G = (V, E)$ be a complete graph, with $V = \{1, \ldots, n\}$ and $w : E \to \mathbb{R}$ a weight function. First observe that the set of Hamilton tours of G is in correspondence with the set of linear orders on V. So, for example, the tour in Figure 3.1 corresponds to the order $(1, 2, 4, 5, 3)$. Actually, distinct orders may represent the

same tour (e.g. $(2, 4, 5, 3, 1)$), but we can fix our attention to those orders having node 1 in first position. It is easy to see that there is a one-to-one correspondence between the Hamilton tours of the complete graph G and the linear orders of its nodes having node 1 in first position. Positions $2, \ldots, n$ may be occupied by any other node. For a given Hamilton tour S represented as an order, the Sarvanov and Doroshko neighborhood $N(S)$ consists in all those orders which can be obtained from S by keeping fixed the nodes in odd position, while letting all even nodes change their position in all possible ways.

Sarvanov and Doroshko neighborhood

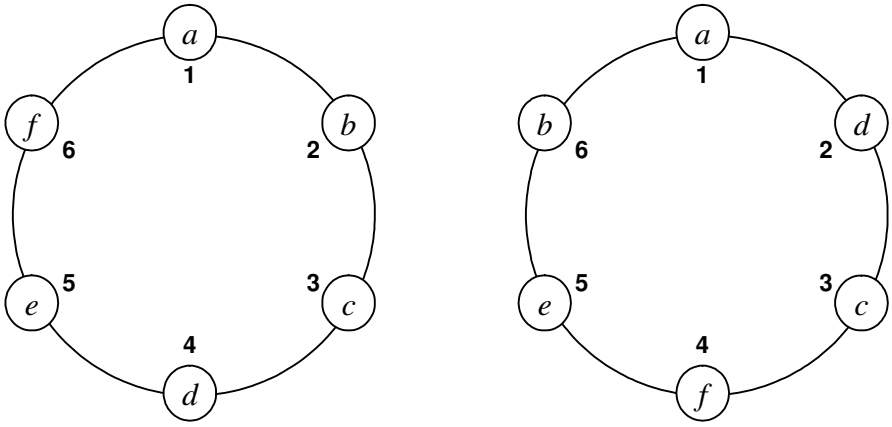

Figure 3.6: Two distinct tours in the Sarvanov and Doroshko neighborhood

Thus, all nodes in even position may appear in a different, but still *even* position in the new Hamilton tour. So, for example, if we consider the following tour on 6 nodes $S_1 = (a, \mathbf{b}, c, \mathbf{d}, e, \mathbf{f})$ (in bold nodes in even position), then its neighborhood will be $N(S) = \{S_1, S_2, S_3, S_4, S_5, S_6\}$, where $S_2 = (a, \mathbf{b}, c, \mathbf{f}, e, \mathbf{d})$, $S_3 = (a, \mathbf{d}, c, \mathbf{b}, e, \mathbf{f})$, $S_4 = (a, \mathbf{d}, c, \mathbf{f}, e, \mathbf{b})$, $S_5 = (a, \mathbf{f}, c, \mathbf{b}, e, \mathbf{d})$, , $S_6 = (a, \mathbf{f}, c, \mathbf{d}, e, \mathbf{b})$. Observe that if in this example node b occupies position 6 in the final order, then arc (e, b) and arc (b, a) will appear in the corresponding Hamilton tour, and the contribution to the overall cost of such two arcs will be $w(e, b) + w(b, a)$. In other words, the cost of assigning each even node to an even position can be easily computed in advance by looking at the odd nodes adjacent to such position, which stay unchanged.

So, assume for the sake of simplicity n to be even, and let $S = (v_1, v_2, \ldots, v_n)$ be a given order. The cost of assigning node v_{2i}, for $i = 1, \ldots, n/2$ to position v_{2k}, for $k = 1, \ldots, n/2$ will be $c_{ik} = w(v_{2k-1}, v_{2i}) + w(v_{2i}, v_{2k+1})$ (where $n+1 = 1$). We have $n/2$ even nodes and $n/2$ even positions and each tour in the neighborhood of S corresponds to a perfect matching between the set of even nodes and the set of even positions. Hence, an optimal solution in the neighborhood of S is simply

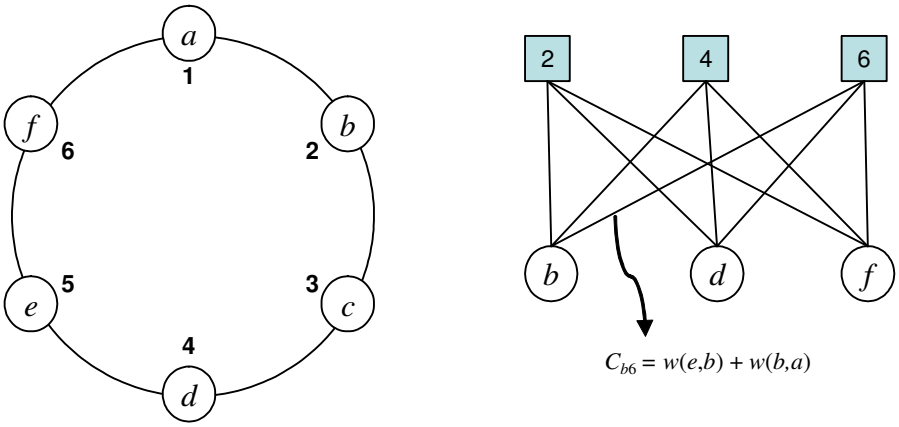

Figure 3.7: A tour and its associated matching problem

a perfect matching M minimizing $c(M)$. Observe that the number of perfect matchings if $n/2!$, but a minimum weight perfect matching can be found very efficiently!

Chapter 4

Exact methods

In this chapter we discuss several methods for actually solving (numerically) combinatorial optimization problems or, equivalently, 0-1 linear programs. This is a very active research area and many interesting techniques are available. The general combinatorial optimization problem is *NP*-hard, so it is quite unlikely that one can find an efficient algorithm (polynomial running time) for this problem. However, the class of combinatorial optimization problems is very large and contains important subclasses with specific properties that algorithms may and should exploit. It is therefore important to have different basic algorithmic principles at hand in order to develop a suitable algorithm for a problem of interest. We shall give a brief presentation of some main techniques that are commonly used for solving integer linear programming and combinatorial optimization problems.

A very good reference book for various methods in integer linear programming and combinatorial optimization is [15]. For algorithms in network flows, see [2].

4.1 Relaxations and branch-and-bound

In Chapter 1 we gave the definition of relaxation of an optimization problem (see Definition 1.2). Recall that a relaxation of an optimization problem is in most cases obtained by enlarging the set of feasible solutions. This is done to get an optimization problem which is easier to solve and thereby obtain a bound on the optimal value of interest. Such bounds were used in Chapter 1 to evaluate the quality of some feasible solution to the problem. In this chapter we will see how these bounds can be used to compute *exact solutions*, that is an optimal solution to the combinatorial optimization problem.

Consider a combinatorial optimization problem (Q) $\max\{w(x) : x \in S\}$, where $S \subseteq \{0,1\}^n$ is the set of feasible solutions to Q. Also, let (R) $\max\{w(x) :$

$x \in T\}$ be a relaxation of (Q) (i.e. $S \subseteq T$), with the property that (R) can be solved efficiently (to optimality) by some exact method. Since we are dealing with combinatorial optimization problems, we may assume that both S and T are bounded. Also, recall that, by definition, we have $v(\mathrm{R}) \geq v(\mathrm{Q})$. Finally, let LB be a lower bound on the optimal value $v(\mathrm{Q})$ to (Q). For example, if we have at hand a feasible solution $\bar{x} \in S$ then we can set $LB = w(\bar{x})$, or we may let $LB = -\infty$ if no such solution is available. Now, since we know how to do it efficiently, we solve (R) instead of (Q). Let x_R be an optimal solution to (R). Several cases can occur and in some of these cases the solution to (R) also provides us with a solution to (Q). In particular, this happens when

(i) (*Infeasibility*): (R) admits no solution, which implies that T is empty, which in turn implies that S is empty and (Q) has no solution.

(ii) (*Optimality*): $x_R \in S$. Then x_R is an optimal solution to (Q). In fact, $x_R \in S$ implies $w(x_R) \leq v(\mathrm{Q})$. Since x_R is optimum to (R), then $w(x_R) = v(\mathrm{R}) \geq v(\mathrm{Q})$.

(iii) (*Value dominance*) $v(\mathrm{R}) = w(x_R) \leq LB$. Since $w(x_R) \geq w(x)$ for all $x \in T$, and $S \subseteq T$, this implies that the solution at hand $\bar{x} \in S$ is already optimal or, in any case, no solution in S can improve over the lower bound LB.

We can then state the following

Proposition 4.1. *Let* (Q) $\max\{w(x) : x \in S\}$ *be a combinatorial optimization problem, let LB be a lower bound on* $v(\mathrm{Q})$ *and let* (R) *be a relaxation of* (Q). *Suppose* (R) *is solved to optimality. Then the solution of* (R) *also provides a solution of* (Q) *whenever:*

(i) (Infeasibility*):* (R) *has no solution.*

(ii) (Optimality*):* (R) *has an optimal solution* x_R *and* $x_R \in S$

(iii) (Value dominance*):* (R) *has an optimal solution and* $v(\mathrm{R}) \leq LB$.

Even if we have at hand a "good" relaxation (R) of (Q), it is often the case that the solution of (R) does not fall into one of the conditions of Lemma 4.1. How can we find an optimal solution to (Q) (or prove that none exists) in this case?

In order to solve difficult (combinatorial) optimization problems, relaxations are often combined with enumerative techniques, based on the so called *divide and conquer* approach.

divide and conquer

Let $S(u), u \in V$ be a partition of S, so these sets are pairwise disjoint and their union equals S. Then we have that

$$v(Q) = \max v(Q(u))$$

where $(Q(u))$ is the restricted problem $\max\{w(x) : x \in S(u)\}$. By solving the restricted problems and comparing we can find an optimal solution to (Q). In

fact, an optimal solution to (Q) belongs to exactly one $S(u)$, for $u \in V$. In any case, the optimal solution $x^*(u) \in S(u)$ to (Q(u)) is also feasible to (Q), since $S(u) \subseteq S$. and it provides us with a lower bound $w(x^*(u))$ on $v(\mathrm{Q})$.

Since $S \subseteq \{0,1\}^n$, a natural way of partitioning S is to fix components to either 0 or 1. For instance, we can let S^0 (S^1) be those vectors in S with $x_1 = 0$ ($x_1 = 1$). Furthermore, one can establish partitions recursively by fixing new variables based on the previous partition. For instance, let $S^{0,0}$ ($S^{0,1}$) be those $x \in S$ with $x_1 = 0$ and $x_2 = 0$ ($x_1 = 0$ and $x_2 = 1$). For simplicity in the presentation we shall assume that there is a predetermined ordering of the variables which is followed when variables are fixed.

enumeration tree

The recursive partitioning may be organized in an *enumeration tree* (or *branching tree*) with nodes corresponding to subsets of S and edges indicating a fixing. The nodes are partitioned into n layers; in the k'th layer the first k variables are fixed. Thus there are at most 2^k nodes in the k'th layer correspond to all possible ways of fixing the k first variables. For each node the subset $S(v)$ of S consists of those vectors in S that satisfy the variable fixing specified in that node. The restricted optimization problem $\max\{w(x) : x \in S(u)\}$ associated with node u will be denoted by $(Q(u))$. The single node v_r in layer 0 is called the *root node* and each of the nodes in the n'th layer are called bottom nodes. When u is a bottom node the problem $(Q(u))$ is trivial as all variables are fixed. The edges go between two consecutive layers. More precisely, a "mother node" on layer k has two adjacent nodes ("children") in layer $k+1$ and they are obtained from the variable fixing in the mother node by fixing x_{k+1} to either 0 or 1. The bottom nodes represent a complete enumeration of all the feasible solutions in the problem (Q).

Even for moderate values of n the enumeration tree is too large for practical purposes. The key idea is that we may only need to consider a small part of this tree in order to solve the problem. For a nonroot node u the tree contains a unique path $p(u)$ between u and the root node. For each nonroot node $w \in p(u)$ we say that u is *below* w or that w is *above* u. Under certain conditions a node u may be *pruned* which means that we need not solve any of the problems (Q(u')) for nodes u' that are below u. This happens when we know that these problems can not improve on the current best solution. The following pruning criteria should be clear.

pruning

Proposition 4.2. *A node u in the enumeration tree may be pruned if one of the following conditions holds.*

(i) (Infeasibility.) $S(u)$ is empty.

(ii) (Optimality.) An optimal solution of (Q(u)) *is known.*

(iii) (Value dominance.) $v(\mathrm{Q(u)}) \leq v(\mathrm{Q})$.

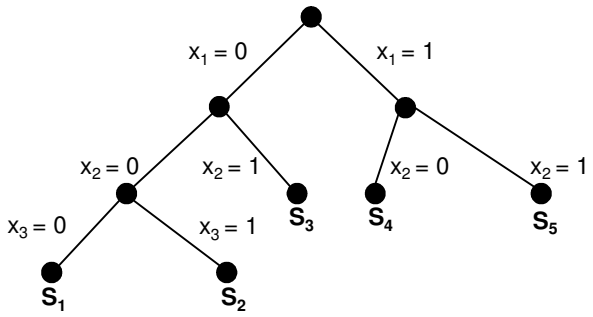

Figure 4.1: A (partial) enumeration tree

Enumeration trees with pruning may be represented by partial binary trees, like the one of Figure 4.1. Note that the leaves of a partial binary tree correspond to the classes of a partition of S, each class being identified by the variable fixing associated to the unique path from the corresponding leave to the root. The tree in Figure 4.1 represents the partition $\{S_1., \ldots, S_5\}$, where $S_1 = \{(0,0,0)\}$, $S_2 = \{(0,0,1)\}$, $S_3 = \{(0,1,0),(0,1,1)\}$, $S_4 = \{(1,0,0),(1,0,1)\}$, $S_5 = \{(1,1,0),(1,1,1)\}$. Solving the original problem (Q) is equivalent to solving problems $(Q_1), \ldots, (Q_5)$, associated to $S_1, \ldots, S_5$. Finally observe that we may let different variables be fixed in a same layer of a partial enumeration tree. An example is given in the Figure 4.2. These more general trees are called *branching trees*.

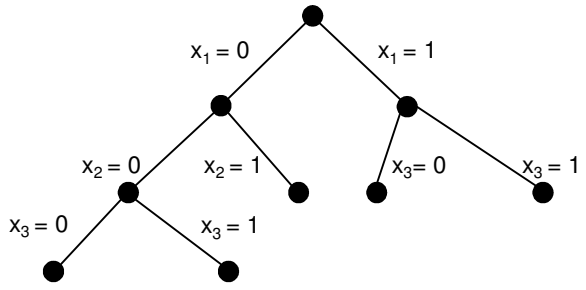

Figure 4.2: A branching tree

One way of deciding if node u can be pruned is therefore to solve the restricted program (Q(u)). This is usually not desirable as (Q(u)) is a difficult (although smaller-dimensional) combinatorial optimization problem. Instead one may solve a relaxation of (Q(u)). This leads to the following pruning criteria. We let (R(u)) denote a relaxation of (Q(u)), and (for simplicity) assume that the objective functions of these problems coincide. The following is a simple consequence of Proposition 4.1:

Proposition 4.3. *A node u in the enumeration tree may be pruned if one of the following conditions holds.*

(i) (Infeasibility) (R(u)) *is infeasible.*

(ii) (Optimality.) We have an optimal solution $\bar{x}$ to (R(u)) *satisfying $\bar{x} \in S(u)$.*

(iii) (Value dominance.) $v(\text{R(u)}) \leq z_L$ where z_L is the objective function for some point in S.

We are mainly interested in the case when all the relaxations $R(u)$ are LP relaxations, i.e., they are obtained by constructing some formulation $P(u)$ for $S(u)$ and solving the corresponding LP $\max\{w^T x : x \in P(u)\}$. Suppose branching node u is obtained from the mother node v by fixing the i'th component to b (where b is either 0 or 1), i.e. $S(u) = \{x \in S(v) : x_i = b\}$. Let $P(v)$ be a formulation for $S(v)$. Then a formulation $P(u)$ for $S(u)$ can be immediately obtained by letting $P(u) = \{x \in P(v) \wedge (x_i = b)\}$, i.e. by including the constraint $x_i = b$ into the linear description of $P(v)$.

We finally observe that, from LP duality, the value $v(R(u))$ may also be found by solving the LP dual $D(u)$ of (u) as we have $v(D(u)) = v(R(u))$ (provided that at least one of the problems is feasible). Therefore, if $D(u)$ is unbounded, then $R(u)$ must be infeasible and the node u may be pruned. Secondly, if we find a feasible solution $\bar{y}$ of $D(u)$ with (dual) objective value which is no greater than z_L, then we may prune due to value dominance (as this implies that $v(D(u)) \leq z_L$).

This leads to an enumerative algorithm based on linear programming. It consists in processing nodes (solving associated optimization problems) in the enumerative tree. We never process a node before all the nodes above that node have been processed. In the algorithm V_n is a list of nodes that remains to be processed. The algorithm is called *branch-and-bound* as we branch in the nodes and determine bounds on the optimal value in each of the nodes.

branch-and-bound

Branch-and-bound algorithm. *Step 1. (Initialization) Let $V_n = \{v_r\}$, $z_L = -\infty$ and $z_U = \infty$.*

Step 2. (Termination.) If $V_n = \emptyset$, the current best solution x^* is optimal; terminate.

Step 3. (Node selection and processing.) Select a node u in V_n and set $V_n := V_n \setminus \{u\}$. Solve the LP relaxation (R(u)). *Let $z(u)$ and $x(u)$ denote the optimal value and an optimal solution to* (R(u)), *respectively.*

Step 4. (Pruning.)

(i) *If* (R(u)) *is infeasible, go to Step 2.*
(ii) *If $x(u) \in S(u)$ and $w^T x(u) > z_L$, update the best solution by setting $x^* = x(u)$ and $z_L = w^T x(u)$ and go to Step 2.*
(iii) *If $z(u) = w^T x(u) \leq z_L$, go to Step 2.*

Step 5. (Branching.) Add two new nodes u_0 and u_1 to V_n each being a child of node u such that $S(u_0)$ and $S(u_1)$ is a partition of $S(u)$. Go to Step 2.

branching variable

Typically, branching is performed on a fractional variable. For instance, if the optimal solution found in node u has two fractional variables x_1, x_4 one selects one of these, say x_4, and introduces the new node u_0 with the additional constraint $x_4 = 0$ and another new node u_1 with the additional constraint $x_4 = 1$. There are other natural ways of introducing new partitions as well, see [15].

Example 4.1. *The linear relaxation associated to the natural formulation for the Expample* 1.1 *is:*

$$\begin{array}{lll} \max & 4x_1 + 5x_2 & \\ & 3x_1 + 4x_2 \leq 6 & \\ & 5x_1 + 2x_2 \leq 6 & \qquad\qquad (\mathrm{R(u)}) \\ & 0 \leq x \leq 1 & \end{array}$$

In order to solve the corresponding 0-1 linear program (with $x \in \{0,1\}^2$) we apply branch-and-bound. Let us represent the initial 0-1 program by node u and let $P(u)$ be the associated formulation (with $S(u) = P(u) \cap \{0,1\}^2$). Initially the list of open problems only contains u, that is $V_n = \{u\}$. At Step 3 we extract u and solve the associated relaxation (R(u)). An optimal solution to the above LP is $x_1(u) = x_2(u) = 0.857$, and $z(u) = 7.7142$. Now observe that none of the conditions of Proposition 4.2 is satisfied and u cannot be pruned. In fact, (R(u)) is feasible, $z(u) < z_L = -\infty$ and $x(u) \notin S(u) = P(u) \cap \{0,1\}^2$ ($x(u)$ has fractional components). Thus, we apply branching, by selecting (for instance) variable x_2 and by generating two new nodes, u_0 and u_1, obtained from u by letting $x_2 = 0$ and $x_2 = 1$, respectively. So, the new list of open problems is now $V_n = \{u_0, u_1\}$. We extract, for instance, u_0. The linear relaxation (R(u_0)) of u_0 is:

$$
\begin{array}{rcll}
\max 4x_1 + 5x_2 & & & \\
3x_1 + 4x_2 & \leq & 6 & \\
5x_1 + 2x_2 & \leq & 6 & \qquad (\mathrm{R}(\mathrm{u}_0)) \\
x_2 & = & 0 & \\
0 \leq x \leq 1 & & &
\end{array}
$$

The optimal solution to (R(u_0)) is $x_1(u_0) = 1$, $x_2(u_0) = 0$, and its value is $z(u_0) = 4$. Since $x(u_0) \in S(u_0)$ (all components are 0 or 1) and $z(u_0) > z_L$ we update the best solution $x^* = x(u_0)$, we set $z_L = 4$ and we prune u_0. Now $V_n = \{u_1\}$: we extract u_1 and solve its relaxation

$$
\begin{array}{rcll}
\max 4x_1 + 5x_2 & & & \\
3x_1 + 4x_2 & \leq & 6 & \\
5x_1 + 2x_2 & \leq & 6 & \qquad (\mathrm{R}(\mathrm{u}_1)) \\
x_2 & = & 1 & \\
0 \leq x \leq 1 & & &
\end{array}
$$

The optimal solution to (R(u_1)) is $x_1(u_1) = 0.667$, $x_2(u_0) = 1$, and its value is $z(u_1) = 7.667$. Since $x(u_1) \notin S(u_1)$ and $z(u_1) > z_L$ we resort to branching. The only fractional variable is x_1, so we branch on x_1 and generate two new nodes u_{10} and u_{11}, corresponding to fixing x_1 to 0 and 1, respectively. Now $V_n = \{u_{10}, u_{11}\}$ and we extract u_{10} and solve its relaxation

$$
\begin{array}{rcll}
\max 4x_1 + 5x_2 & & & \\
3x_1 + 4x_2 & \leq & 6 & \\
5x_1 + 2x_2 & \leq & 6 & \qquad (\mathrm{R}(\mathrm{u}_{10})) \\
x_2 & = & 1 & \\
x_1 & = & 0 & \\
0 \leq x \leq 1 & & &
\end{array}
$$

The optimal solution to (R(u_{10})) is $x_1(u_{10}) = 0$, $x_2(u_{10}) = 1$, and its value is $z(u_{10}) = 5$. Since $x(u_{10}) \in S(u_{10})$ and $z(u_{10}) = 5 > 4 = z_L$ we let the best solution to be $x^* = x(u_{10})$, $z_L = 5$ and we prune u_{10}. Now we extract u_{11} from V_n, (R(u_{11})) is infeasible, and u_{11} is pruned. Finally V_n is empty and the algorithm

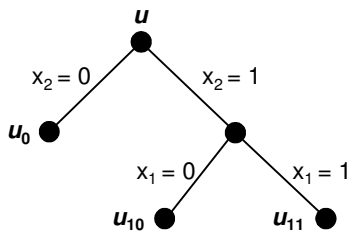

Figure 4.3: The (partial) branch-and-bound tree of Example 4.1

terminates. The partial branching tree associated to the above application of branch-and-bound is shown in Figure 4.3.

Two main issues in the development of branch-and-bound algorithms are *node selection* and *variable selection*.

Whenever we have solved the relaxation $R(u)$, and we do not terminate, we have to select the next node to be processed (also called the next active node). In this node selection problem several strategies exist, some are based on a priori rules and others are adaptive (depending on the calculations). For instance, a common strategy is *breadth-first-search plus backtracking* where one always chooses the next node as a child node and backtracks if the node is pruned. In the variable selection problem one decides how to make the partitioning that determines the (two or more) children problem. Empirically one knows that this choice affects the over-all speed of the algorithm, but still it is hard to find good rules for selecting "critical variables". A useful strategy is to predetermine some ordering of the variables based on the coupling in the constraints. For instance, the variables may fall into two classes such that fixing all the variables in the first class makes the remaining ones integral. In such a case it makes sense to branch on fractional variables in the first class whenever possible. Other possible techniques are discussed in [15].

One crucial point is the strength of the LP relaxations, that is the quality of the formulations and of the corresponding bounds. If the LP (optimal value) bounds are too far away from the optimal value one cannot prune the enumeration tree and the algorithm becomes slow, often too slow for all practical purposes, see

Exercise 4.2. So, we are interested in finding good formulations for the set S of 0-1 solutions of our combinatorial optimization problem. Following the discussion of Chapter 1, we would like to optimize over $\text{conv}(S)$, but typically we need to content ourselves with much weaker formulations, such as the natural one. However, the initial formulation can be significantly strengthen in each node of the branching tree.

cutting planes

Suppose so we have solved the linear relaxation (R(u)) associated to the current formulation $P(u)$, but node u cannot be fathomed. Now, instead of proceeding with branching we may try to strengthen the current formulation. This is done by invoking a separation oracle which tries to find an inequality belonging to a stronger formulation which is violated by the current solution $x(u)$. This inequality is called a *cutting plane*, since it can be seen as a hyperplane (or *cut*) separating $x(u)$ from $\text{conv}(S(u))$. Typically, such a cut belongs to some predefined class of inequalities and in this case, we are within the so called *Template Paradigm*. But we may also look for general inequalities and we discuss an example later in this chapter. The method which combines branch-and-bound and cutting planes is called *branch-and-cut*.

4.2 Finding additional valid inequalities

We discuss some methods for finding classes of valid inequalities for a set of integral points or simply 0-1 points. Ideally we would like to have methods for going from a polyhedron P (the initial formulation for a set of solution $S = P \cap \{0,1\}^n$) to its integer hull P_I (or $\text{conv}(S)$). We shall mention a theoretical result which says that this is indeed possible although the construction is far from practical. Thus we shall also discuss other general techniques that are applicable to any integer linear programming problem.

valid inequality

First we fix some terminology. Let S be some subset of $\mathbb{R}^n$ and let $a^T x \leq \alpha$ be an inequality with $a \neq 0$. We say that $a^T x \leq \alpha$ is *valid* for S if $S \subseteq H_{\leq}(a, \alpha) = \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$, i.e., each point of S satisfies the inequality in question. We use the notation $a_{i,.}$ to denote the i'th row of $A \in \mathbb{R}^{m,n}$ viewed as a column vector. Similarly, $a_{.,j}$ denoted the j'th column of A.

The Chvátal-Gomory procedure.

Let $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ be a given polyhedron in $\mathbb{R}^n$ with A, b rational. We are interested in the problem of finding valid inequalities for the integer hull P_I. Clearly, each inequality in $Ax \leq b$ is valid for P_I, and the purpose is to provide

methods for finding additional ones. The basic idea to be discussed is based on the simple fact that if an integral number x satisfies $x \leq \alpha$, then it also satisfies $x \leq \lfloor \alpha \rfloor$. This strengthening of the inequality is called *integer rounding*.

Geometrically, as P_I is spanned by the integer points in P, what we need is some procedure for "pushing" a hyperplane defined by an inequality in the system $Ax \leq b$ as far as possible towards P_I. Ideally, we would like to push until we meet a point in P_I, but in higher dimensions this may not be so easily achieved. What we can do instead is to push the hyperplane until an integer point is met (although this point is outside P). For instance, if $x_1 + 2x_2 \leq 10/3$ is a valid inequality for P, then the "rounded inequality" $x_1 + 2x_2 \leq \lfloor 10/3 \rfloor = 3$ is also valid for P_I. Note that none of the parallel hyperplanes defined by the inequalities $x_1 + 2x_2 \leq \gamma$ contain integral points for $3 < \gamma \leq 10/3$ (prove this!). Two simple algebraic facts are that (i) multiplying a valid inequality by a positive number gives another (equivalent) valid inequality, and that (ii) the sum of valid inequalities is again a valid inequality. We remark that these properties may be expressed in terms of convexity as follows. Let $\bar{P}$ be the subset of $\mathbb{R}^{n+1}$ consisting of points (a, α) for which $a^T x \leq \alpha$ is a valid inequality for P. The mentioned algebraic properties of valid inequalities simply mean that $\bar{P}$ is a convex cone in $\mathbb{R}^{n+1}$. If we combine the rounding idea with the cone property of the valid inequalities we get the following procedure for finding a valid inequality. For simplicity, we assume that $P \subseteq \mathbb{R}^n_+$, i.e., that $x_j \geq 0$ for $j \leq n$ are valid inequalities for P. Multiply the i'th inequality $a^T_{i,\cdot} x \leq b_i$ by λ_i for each $i \leq m$ and sum all the inequalities. This gives the new inequality $(\sum_{i \leq m} \lambda_i a^T_{i,\cdot})x \leq \sum_{i \leq m} \lambda_i b_i$. If we let $\lambda = (\lambda_1, \ldots, \lambda_m)^T$ this new inequality is $(\sum_{j \leq n} \lambda^T a_{\cdot,j}) x_j \leq \lambda^T b$. This inequality is redundant (as it is implied by $Ax \leq b$), but now we may apply integer rounding. Thus we see that $\sum_{j \leq n} \lfloor \lambda^T a_{\cdot,j} \rfloor x \leq \lambda^T b$ is valid for P as we assumed that each $x \in P$ satisfies $x \geq 0$. But if we insert an integral point x in P on the left-hand-side of this inequality we get an integer (all numbers are then integers). Thus we may round the right-hand-side down to the nearest integer and still have a valid inequality, namely

$$\sum_{j \leq n} \lfloor \lambda^T a_{\cdot,j} \rfloor x \leq \lfloor \lambda^T b \rfloor. \tag{4.1}$$

The procedure leading to (4.1) is called the *Chvátal-Gomory procedure* and we also call (4.1) a *Chvátal-Gomory inequality*.

Example 4.1 *(continued).* Consider the original formulation $P(u)$ of Example 4.1.

$$
\begin{aligned}
\max\, & 4x_1 + 5x_2 & & & \\
& 3x_1 + 4x_2 & \leq & 6 & \\
& 5x_1 + 2x_2 & \leq & 6 & \qquad \text{(R(u))} \\
& -x_1 & \leq & 0 & \\
& -x_2 & \leq & 0 & \\
& x_1 & \leq & 1 & \\
& x_2 & \leq & 1 &
\end{aligned}
$$

Consider the following vector $\lambda = (2/10, 1/10, 0, 0, 0, 0)$ which corresponds to summing up the first and the second constraint multiplied by 2/10 and 1/10, respectively. We obtain the following inequality, which is valid for the feasible region $P(u)$ of (R(u)):

$$11/10x_1 + x_2 \leq 18/10$$

If we round down all the coefficients we obtain the following inequality, which is valid for $S(u) = P(u) \cap \{0, 1\}^2$ but not for $P(u)$:

$$x_1 + x_2 \leq 1$$

Interestingly, the above inequality plus the box constraints of $P(u)$ define the convex hull of $S(u)$.

Note that the Chvátal-Gomory procedure may be applied repeatedly and thereby generating gradually larger classes of valid inequalities. How far may we reach by doing this? A remarkable fact is that *every* valid inequality for P may be generated in this way (possibly by increasing the right-hand-side) provided that suitably many repetitions are taken. More specifically, for each linear system $Ax \leq b$ defining the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ we can find a finite family of linear systems $A^{(k)}x \leq b^{(k)}$ for $k = 0, \ldots, t$ such that (i) $A^{(0)} = A$, $b^{(0)} = b$, (ii) each inequality in $A^{(k)}x \leq b^{(k)}$ is derived from $A^{(k-1)}x \leq b^{(k-1)}$ using the Chvátal-Gomory procedure for $k = 1, \ldots, t$, and (iii) $P_I = \{x \in \mathbb{R}^n : A^{(t)}x \leq b^{(t)}\}$. For a proof of this result, see [20] or [15].

mathching polytope

As an example we consider the matching problem. A *matching* in a graph $G = (V, E)$ is a subset M of the edge set E such that $d_{(V,M)}(v) \leq 1$ for each node $v \in V$, i.e., each node is the endnode of at most one edge in M. One can check

that the set of 0-1 vectors satisfying $0 \leq x \leq 1$ and $x(\delta(v)) \leq 1$ for each $v \in V$ coincides with the set of incidence vectors to matchings in G. Let $Ax \leq b$ denote this linear system. The polyhedron $P = \{x \in R^V : Ax \leq b$ is a formulation for the set of incidence vectors of the matchings in G, and is called the *fractional matching polytope*. The *matching polytope* is defined as the convex hull of the incidence vectors of matchings. Let $S \subset V$ consist of an odd number of nodes, say $|S| = 2k + 1$. Then the inequality $x(E[S]) \leq k$ is clearly valid for P_I as each matching contains no more than k pairs of nodes in S. Thus, the validity is due to a simple combinatorial argument. However, this inequality may also be obtained using the Chvátal-Gomory procedure on the original system $Ax \leq b$. Consider the inequalities $x(\delta(v)) \leq 1$ for $v \in S$ and the inequalities $-x_e \leq 0$ for $e \in \delta(S)$. If we multiply each of these inequalities by $1/2$ and add them together, we get the (valid) inequality $x(E[S]) \leq k+1/2$. By integer rounding we get the desired inequality $x(E[S]) \leq k$ which proves that this inequality may be obtained by the Chvátal-Gomory procedure applied to the system consisting of the degree inequalities and simple bounds. The matching polytope is completely described by the degree inequalities, simple bounds and the odd set inequalities $x(E[S]) \leq \lfloor |S|/2 \rfloor$ for $S \subseteq V$ and S odd. Thus, all the facet defining inequalities for the matching polytope are obtained by adding "one round of cutting planes" and we say that the original polyhedron P has Chvátal-Gomory-rank 1.

A feature of the Chvátal-Gomory procedure is that it may be affected by scaling of the inequalities. For instance, if $P = \{x \in \mathbb{R} : x \leq 3/2\}$ then $P_I = \{x \in \mathbb{R} : x \leq 1\}$ and the inequality $x \leq 1$ is obtained by integer rounding from $x \leq 3/2$. However, P is also the solution set of the (scaled) inequality $2x \leq 3$, and rounding directly on this inequality does not change the inequality. From this we realize that integer rounding should be preceded by a proper scaling of the inequality, i.e., dividing by the greatest common divisor of all the numbers involved. For a single inequality this produces the integer hull as the next result says.

Proposition 4.4. *Let $P = \{x \in \mathbb{R}^n : \sum_{j=1}^n a_j x_j \leq \alpha\}$ where all the a_j's are integers. Let d be the greatest common divisor of $a_1, \ldots, a_n$. Then $P_I = \{x \in \mathbb{R}^n : \sum_{j=1}^n (a_j/d) x_j \leq \lfloor \alpha/d \rfloor\}$.*

Boolean implications.

knapsack problem

Sometimes one can find new inequalities by detecting logical (boolean) implications of one or more constraints of the original system.

The problem $\max \{\sum_{j=1}^n c_j x_j : \sum_{j=1}^n a_j x_j \leq b,\ 0 \leq x \leq 1\}$ where all the data are positive integers, is called the *knapsack problem*. Let $P = \{x \in \mathbb{R}^n : \sum_{j=1}^n a_j x_j \leq b,\ 0 \leq x \leq 1\}$, $S = P \cap \{0,1\}^n$ and $\text{conv}(S)$ is the *knapsack polytope*. As a

specific example let $n = 3$, $a_1 = 3$, $a_2 = 3$, $a_3 = 2$ and $b = 7$. Let $C = \{1, 2, 3\}$ and note that $a(C) = \sum_{j \in C} a_j = 8 > b$. We call C a *dependent set* or *cover*. Since $a(C) > b$, no feasible integral solution in P can have all variables in C equal to 1. Thus $x(C) \leq |C| - 1$ is a valid inequality for P_I which is often called a *cover inequality*. These inequalities have proved to be very useful for solving several different combinatorial optimization problems since cover inequalities may be derived for individual constraints in the integer linear programming formulation. We also remark that for the polyhedron P above (the linear relaxation of the knapsack problem) all the vertices may be described in a simple way, see Problem 4.1.

cover inequality

Note that *any* linear inequality in 0-1 variables may be transformed to the situation just treated. A general linear inequality with rational data may be written (after suitable scaling) $\sum_{j \in J_1} a_j x_j + \sum_{j \in J_2} a_j x_j \leq b$ with a_j, $j \in J_1$ positive integers and a_j, $j \in J_2$ negative integers (j's with $a_j = 0$ are not of interest for the analysis). We make the affine transformation $z_j = 1 - x_j$ for $j \in J_2$, and the transformed inequality is $\sum_{j \in J_1} a_j x_j + \sum_{j \in J_2} (-a_j) z_j \leq b - \sum_{j \in J_2} a_j$. Here all coefficients are positive, so we may establish (e.g.) cover inequalities as above and finally transform these back into new valid inequalities for the original problem.

variable upper bound

Another type of constraint that is often met in applications is

$$\begin{array}{ll} \text{(i)} & \sum_{j \in N} y_j \leq nx; \\ \text{(ii)} & 0 \leq x \leq 1,\ 0 \leq y \leq 1; \\ \text{(iii)} & x \in \{0, 1\}. \end{array} \tag{4.2}$$

The logical contents of the constraints is that $x = 1$ whenever the sum of the continuous variables $y_j, j \in N$ is positive. Alternatively, all $y_j, j \in N$ must be 0 whenever $x = 0$. This type of constraints are called *variable upper bounds*. However, this means that all the inequalities $y_j \leq x$ for $j \in N$ are also valid for the solution set of (4.2). By adding these new constraints we cut off fractional solutions (from the continuous relaxation of (4.2)) like e.g., $y_1 = 1$, $y_j = 0$ for $j \in N \setminus \{1\}$, $x = 1/n$ since the inequality $y_1 \leq x$ is violated. For more about related mixed integer sets like the variable upper-bound flow models, confer [15].

variable upper bound

Combinatorial implications.

For combinatorial polyhedra, i.e., polyhedra with vertices corresponding to some class of combinatorial objects, one may find valid inequalities by exploiting these combinatorial properties. The example with the matching polytope given in the previous paragraph fits into this framework. Here we give some other examples.

set covering problem

First, we consider the *set covering problem* (see e.g., [4]). This problem is of relevance to many applications. For instance, in airline crew scheduling each flight must be covered; in allocating student classes to rooms, each class must get a room, or in network design a number of capacity "bottlenecks" (like cuts) must be covered. Let I and J be the color classes of a bipartite graph G, so each edge in the edge set E joins some node in I to some node in J. Let c_j for $j \in J$ be given nonnegative weights. By a *cover* we understand a subset S of J such that each node in I is adjacent to at least one node in S. (Of course, we assume that G allows this to happen, i.e., each node in I is adjacent to some node in J). The set covering problem is to find a cover of minimum weight, where the weight $w(S)$ of a cover S is defined as $w(S) = \sum_{j \in S} w_j$. This problem is *NP*-hard. The polyhedral approach to this problem may start by introducing the *set covering polytope* P_{SC} as the convex hull of vectors χ^S where S is a cover. An integer linear programming formulation of the set covering problem is obtained by letting x_j indicate whether node j is in the cover to be determined. For $i \in I$ we let $\Gamma(i)$ denote the set of nodes in J that are adjacent to i.

$$\begin{array}{lll} \text{minimize} & \sum_{j \in J} c_j x_j & \\ \text{subject to} & & \\ \text{(i)} & \sum_{j \in \Gamma(i)} x_j \geq 1 & \text{for all } i \in I; \\ \text{(ii)} & 0 \leq x_j \leq 1 & \text{for all } j \in J; \\ \text{(iii)} & x \text{ is binary.} & \end{array} \tag{4.3}$$

Thus the constraint (4.3)(i) assures that each node in I is covered. Let P be the solution set of (4.3)(i)–(ii), so the integer hull P_I of this polytope is precisely the set covering polytope P_{SC}. Due to the hardness of the set covering problem, it is too ambitious to find a complete linear description of P_I. However, in order to numerically solve practical set covering problems one may need to find some of these inequalities and add them to the description of P. In other words, the LP relaxation using P may give poor lower bounds on the true optimal value of the set covering instance of interest. For example, consider a graph with nodes $I = \{i_1, i_2, i_3\}$ and $J = \{j_1, j_2, j_3\}$, and with the six edges $[i_1, j_1], [i_1, j_2], [i_2, j_2], [i_2, j_3], [i_3, j_3], [i_3, j_1]$. Note that each node in J covers two consecutive nodes in I. Assume that the objective function is $c = (1, 1, 1)$. Then an (in fact, *the*) optimal solution of the LP relaxation is $\bar{x} = (1/2, 1/2, 1/2)$ with objective value $3/2$. The optimal value of the integer program, and therefore the set covering problem, is 2. Thus, some valid inequality for P_{SC} is required to cut off this fractional vertex of P. Such an inequality may be deduced from a simple combinatorial argument: no single node in J can cover all the nodes in I, therefore at least two nodes in J must be chosen. Thus the inequality $x_1 + x_2 + x_3 \geq 2$ is valid for P_{SC} (as it holds for all vertices and then, by convexity, for all points of P_{SC}). Also, it is violated by $\bar{x}$. If we add

this inequality to our current linear program and reoptimize we get an optimal integer solution, say $(1, 1, 0)$. The inequality we just identified actually belongs to a large class of valid, and often non-redundant, inequalities for set covering polytopes: the *odd cover inequalities* . It may seem that we only get a few inequalities in this way, but for a given graph G there may be many subgraphs that are isomorphic to the one of our example, and each of these produce several valid inequalities called *lifted odd cover inequalities*. The procedure involved is called *lifting* and makes it possible to find facets of a higher dimensional polytope via facets of lower dimensional ones (namely projections of the polytope of interest), see [15].

node packing problem

As another example we consider the *node packing problem*. A node packing (*independent set, stable set*) in a graph $G = (V, E)$ is a subset S of the nodes such that no pair of nodes is adjacent. The *node packing polytope* (see [10]) is the convex hull P_{NP} of the incidence vectors of node packings in G. Note that this polytope depends on G. A binary vector $x \in \mathbb{R}^V$ is the incidence vector of a node packing iff $x_u + x_v \leq 1$ for each $[u, v] \in E$. Thus $P_{NP} = P_I$ where $P = \{x \in \mathbb{R}^V : 0 \leq x \leq 1,\ x_u + x_v \leq 1 \text{ for each } [u, v] \in E\}$. It can be shown that $P = P_I$ iff the graph G is bipartite. Thus, for general graphs further inequalities are needed to define the node packing polytope. For instance, consider a clique which is a complete subgraph, i.e., a subset V_0 of V such that $[u, v] \in E$ for all distinct $u, v \in V_0$. Clearly any node packing contains at most one node in such a clique, so the *clique inequality* $x(V_0) \leq 1$ is valid for P_{NP}. Note that this inequality is stronger than the inequalities $x_u + x_v \leq 1$ for $u, v \in V_0$. This means that each of these inequalities is implied by the clique inequality and the nonnegativity constraints. Next, consider an odd cycle C with, say, $2k + 1$ nodes. Then at most every second node can lie in a node packing, so exploiting the parity property we get the valid inequality $x(C) \leq k$. A major research topic in polyhedral combinatorics is the study of those graphs for which the clique constraints and the nonnegativity constraints are sufficient to describe the node packing polytope, see [15], [14]. Such graphs are called *perfect graphs*.

4.3 Lagrangian relaxation

In Section 4.1 we discussed relaxations of optimization problems in a general setting. We here consider one specific type of relaxation that has turned out to be of great value in finding near-optimal solutions to (several) combinatorial optimization problems. The idea in Lagrangian relaxation is to exploit the underlying structure of an optimization problem in order to produce bounds on the optimal value.

Consider the 0-1 linear programming problem with feasible set $S = P \cap Z^n$ where $P = \{x \in \mathbb{R}^n : Ax \leq b,\ 0 \leq x \leq 1\}$ and $A \in \mathbb{R}^{m,n}$. (The following development also works more generally, in fact for $S = P \cap X$ where X is any subset of $\mathbb{R}^n$). Assume that the system $Ax \leq b$ is split into two subsystems $A^1x \leq b^1$ and $A^2x \leq b^2$ where A^i has m^i rows and $m^1 + m^2 = m$. One can think of $A^2x \leq b^2$ as "complicating constraints" in the sense that if they were dropped an easier problem would be obtained. Thus we have $P = \{x \in \mathbb{R}^n : A^1x \leq b^1,\ A^2x \leq b^2,\ 0 \leq x \leq 1\}$. The 0-1 linear programming problem (Q) may be written as follows.

$$\begin{array}{ll} \max & c^Tx \\ \text{subject to} & \\ \text{(i)} & A^1x \leq b^1; \\ \text{(ii)} & A^2x \leq b^2; \\ \text{(iii)} & 0 \leq x \leq 1; \\ \text{(iii)} & x \text{ is integral.} \end{array} \tag{4.4}$$

The purpose of this constraint splitting is to open up for an associated and simpler 0-1 LP problem where the constraints $A^2x \leq b^2$ have been moved to the objective function with penalties. We consider the following problem $LR(\lambda)$

$$\begin{array}{ll} \max & c^Tx + \lambda^T(b^2 - A^2x) \\ \text{subject to} & \\ \text{(i)} & A^1x \leq b^1; \\ \text{(ii)} & 0 \leq x \leq 1; \\ \text{(iii)} & x \text{ is integral.} \end{array} \tag{4.5}$$

where $\lambda = (\lambda_1, \ldots, \lambda_{m^2})$ consists of nonnegative weights or "penalties", usually called the *Lagrangian multipliers*. Thus, in $LR(\lambda)$ a feasible point $\bar{x}$ may violate a constraint $a_i^Tx \leq b_i$ in $A^2x \leq b^2$ but this increases the objective function by the amount of $\lambda_i(b_i - a_i^T\bar{x})$. On the negative side, we see that we get an "award" in the objective by satisfying an inequality strictly. This is an unavoidable problem when we want to maintain a linear objective function.

We call the problem $LR(\lambda)$ the *Lagrangian relaxation* (or *Lagrangian subproblem*) w.r.t. the constraints $A^2x \leq b^2$. As the name indicates this Lagrangian relaxation is really a relaxation of (4.4) for any $\lambda \in \mathbb{R}^m_+$. To see this we note that the feasible region of the Lagrangian relaxation contains the feasible region of the original problem. In addition, if $\bar{x}$ is feasible in (4.4), then, in particular, we have $A^2\bar{x} \leq b^2$ and therefore also $c^T\bar{x} + \lambda^T(b^2 - A^2\bar{x}) \geq c^Tx$ as λ is nonnegative. Thus we obtain an upper bound on the optimal value of interest:

$$v(Q) \leq v(LR(\lambda)).$$

Since this holds for all $\lambda \in \mathbb{R}^n_+$, the best upper bound obtained in this way is given by solving the so-called *Lagrangian dual problem* (LD) (w.r.t. $A^2x \leq b^2$)

$$\min\{v(LR(\lambda)) : \lambda \geq 0\} \tag{4.6}$$

and we get the important inequalities

$$v(Q) \leq v(LD) \leq v(LR(\lambda)) \text{ for all } \lambda \in \mathbb{R}^n_+. \tag{4.7}$$

The Lagrangian dual problem may be viewed as a nondifferentiable convex minimization problem as $v(LR(\lambda))$ is a piecewise linear and convex function (it is the pointwise minimum of a finite number of affine functions). Algoritmically one tries to solve the Lagrangian dual problem by some kind of multiplier adjustment technique. The basic principle is to adjust the multiplier according to the current optimal solution x. If x violates the constraint, the penalty (multiplier) is increased, but if x satisfies the constraint, the penalty is decreased. Different ideas are used for deciding *how much* these adjustments should be, and for this good stategies are problem dependent. For a discussion of one such general technique, called the *subgradient method*, see [15].

We consider an application which illustrates the idea of Lagrangian relaxation.

The *degree-constrained spanning tree problem* (DCST) is to find a minimum weight spanning tree satisfying given degree constraints. More specifically, let w be a nonnegative weight function defined on the edges of a graph $G = (V, E)$ and let b_v for $v \in V$ be given positive integers. We want to find a spanning tree T satisfying the degree constraints $d_T(v) \leq b_v$ for each $v \in V$ and with minimum total weight $w(T) = \sum_{e \in T} w_e$. (Of course, this problem is infeasible if the b_v's are "too small"). This problem is known to be *NP*-hard, see [8]. But we know that the spanning tree problem is tractable, i.e., polynomial, and this can be exploited as follows. It is possible to write down a linear system $A^1x \leq b^1$ with 0-1 solutions that correspond to the incidence vectors of sets $F \subseteq E$ such that (V, F) contains a spanning tree. Finding such a system is left as an exercise, but here we only need to know that the system exists. Then our problem (DCST) may be written as (4.4) with $c = -w$ and the system $A^2x \leq b^2$ being

$$x(\delta(v)) \leq b_v \quad \text{for all } v \in V. \tag{4.8}$$

The Lagrangian relaxation w.r.t. the degree constraints (4.8) is essentially a spanning tree problem. The objective function to be minimized in this problem is

$$w^T x + \sum_{v \in V} \lambda_v (x(\delta(v)) - b_v).$$

This means that the weight of a spanning tree T becomes (use $x = \chi^T$)

$$-\sum_{v\in V} \lambda_v b_v + \sum_{[u,v]\in T} (w_{uv} + \lambda_u + \lambda_v).$$

This objective will therefore tend to give spanning trees having low degrees. The Lagrangian relaxation can for each λ be solved by, e.g., Kruskal's algorithm. Thus, combined with a suitable multiplier technique we can solve the Lagrangian dual and obtain a lower bound on the optimal value of the DCST problem. Also, if we are lucky and find an optimal spanning tree in the final Lagrangian relaxation which satisfies all the degree constraints (4.8), then this solution is also an optimal solution of (4.8). Otherwise, one usually constructs a feasible spanning tree solution by some kind of heuristic method based on the last subproblem. This also produces a bound on the optimality gap.

We return to the general theory of Lagrangian relaxation. Consider again the Lagrangian relaxation w.r.t. $A^2x \leq b^2$ given in (4.5). The objective function $c^Tx + \lambda^T(b^2 - A^2x) = (c^T - \lambda^T A^2)x + \lambda^T b^2$ is an affine function of x, i.e., a linear function $c(\lambda)x$ (with $c(\lambda) := c^T - \lambda^T A^2$) plus some constant. Since the constant may be removed from the optimization, the problem (4.5) consists in maximizing a linear function over the 0-1 vectors in the polyhedron defined by $A^1x \leq b^1$. As discussed in the introduction to this chapter we may convexify such a problem and obtain an equivalent LP problem

$$\max\{c(\lambda)^T x : x \in P_I^1\}$$

where P_I^1 is the integer hull of the polyhedron $P^1 = \{x \in \mathbb{R}^n : A^1x \leq b^1,\ 0 \leq x \leq 1\}$. Thus the Lagrangian relaxation corresponds to "integralization" with respect to the system $A^1x \leq b^1,\ 0 \leq x \leq 1$ and translating the objective function with some linear combination of the row vectors in A^2. To proceed, it follows from Motzkin's theorem that $P_I^1 = \text{conv}(\{x^k : k \in K\})$ where x^k, $k \in K$ is a finite set of vectors in $\mathbb{R}^n$; these are the vertices of P_I^1. Therefore we obtain

$$\begin{aligned}
&v(LD) = \min\{v(LR(\lambda)) : \lambda \geq 0\} = \\
&\min_{\lambda\geq 0}[\max_{x\in P_I^1}(c^T - \lambda^T A^2)x + \lambda^T b^2] = \\
&\min_{\lambda\geq 0}[\max_{k\in K}(c^T - \lambda^T A^2)x^k + \lambda^T b^2] = \\
&\min_{\lambda\geq 0}[\min\{\eta : \eta \geq (c^T - \lambda^T A^2)x^k + \lambda^T b^2, \text{ for all } k \in K\}] = \\
&\min\{\eta : \lambda \geq 0,\ \eta + \lambda^T(A^2x^k - b^2) \geq c^T x^k, \text{ for all } k \in K\} = \\
&\max\{c^T \sum_{k\in K} \mu^k x^k : A^2 \sum_{k\in K} \mu^k x^k \leq b^2 \sum_{k\in K} \mu^k x^k; \sum_{k\in K} \mu^k = 1; \mu \geq 0\} = \\
&\max\{c^T x : A^2 x \leq b^2,\ x \in P_I^1\}.
\end{aligned}$$

The second last equality was obtained using linear programming duality. Note also the transformation used for converting the inner maximization problem into an LP problem of minimizing an upper bound. We have therefore shown the following result.

Theorem 4.5. *The Lagrangian dual problem (4.6) may be viewed as the dual of the LP problem* $\max\{c^T x : A^2 x \leq b^2,\ x \in P_I^1\}$. *In particular, the optimal values of these problems coincide.*

This result is the main result on Lagrangian relaxation. It says that the bound obtained from solving the Lagrangian dual equals the one obtained by the LP problem with feasible set based on integralization only w.r.t. the constraints that are not relaxed. Define the three polytopes

$$\begin{array}{lcl} P_I^{1,2} & := & \{x \in \mathbb{R}^n : 0 \leq x \leq 1,\ A^1 x \leq b^1,\ A^2 x \leq b^2\}_I; \\ (P_I^1)^2 & := & \{x \in \mathbb{R}^n : 0 \leq x \leq 1,\ A^1 x \leq b^1\}_I \cap \{x \in \mathbb{R}^n : A^2 x \leq b^2\}; \\ P^{1,2} & := & \{x \in \mathbb{R}^n : 0 \leq x \leq 1,\ A^1 x \leq b^1,\ A^2 x \leq b^2\}. \end{array} \tag{4.9}$$

Maximizing $c^T x$ over $P_I^{1,2}$ correponds to the original integer program (more precisely, transformed into an LP); maximizing over $(P_I^1)^2$ corresponds to solving the Lagrangian dual and, finally, maximizing over $P^{1,2}$ is simply the LP relaxation of the integer program (4.4). Let LP denote the last program. Since we have

$$P_I^{1,2} \subseteq (P_I^1)^2 \subseteq P^{1,2}$$

we get the following ordering of the optimal values in these optimization problems

$$\begin{array}{l} v(Q) = \max\{c^T x : P_I^{1,2}\} \leq \\ \max\{c^T x : (P_I^1)^2\} = v(LD) \leq \\ \max\{c^T x : P^{1,2}\} = v(LP). \end{array}$$

This means that the Lagrangian bound may improve on the bound coming from the LP relaxation. Note, however, an important consequence of Theorem 4.5 concerning the bounds.

Corollary 4.6. *If the polyhedron* P^1 *is integral, i.e., has integral vertices, then* $v(LD) = v(LP)$.

Thus, if integrality may be dropped in the Lagrangian subproblems (i.e., $\{x \in \mathbb{R}^n : A^1 x \leq b^1\}$ is integral), then we will not improve compared to the bound obtained by solving the original LP relaxation. Usually, in such cases, Lagrangian relaxation is not used unless it is viewed as more practical than solving the LP.

However, if the polyhedron is not integral, then, depending on the objective function, the value of the Lagrangian dual will improve on the LP relaxation bound. Consequently, this should be taken into account when deciding on the splitting on the constraints, so as to make the Lagrangian subproblems "simple, but not too simple".

4.4 The Traveling Salesman Problem

In order to exemplify some of the principles and methods presented above, we discuss the Traveling Salesman Problem (TSP) in this section. Our presentation is very brief. A more detailed presentation is given in [15] and, of course, in the recent "TSP-book" [1].

The TSP has been studied a lot during the last 50 years by mathematicians and computer scientists. The problem is of interest in certain real-world applications, like vehicle routing and computer chip production, but it has also an attraction from a theoretical point of view. One reason is that it is easy to formulate, but difficult to solve!

The TSP problem is to find a shortest trip through a given set of cities. More precisely, we have given an undirected graph $G = (V, E)$ with nonnegative weights (costs) on the edges: c_e, $e \in E$. A *tour* is a Hamilton cycle, i.e., a cycle going through all the nodes of G and passing through each node exactly once. The length of a tour is the sum of the weights of its edges. The problem is to find a shortest tour. The TSP is *NP*-hard, and even deciding if a graph contains a Hamilton tour is *NP*-complete. Even worse, the problem is also hard to approximate as it is *NP*-hard to solve the TSP with any given performance guarantee. That is, for *any* given positive number r it is "hard" to find a tour of length at most $(1 + r)$ times the optimal length. However, some special cases of the TSP are polynomially solvable, see [13]. Many heuristics have been developed for the TSP. Some of them have been discussed in Chapter 3.

Applications. There is a large number of different applications of the TSP to real-life problems. Some of them arise quite naturally, others are maybe a bit more surprising. We give a short list of such applications, far from being exhaustive. Most of the examples are taken from [1].

- *Vehicle routing.* This is probably the most immediate application. Several organizations, such as schools, municipalities, large companies, etc, need to manage fleets of vehicles delivering and picking up passengers or

goods in the local area. Examples are managing school-buses, postal services, garbage collection, etc. In this type of problems several vehicles must be assigned a number of locations to reach, starting from a common depot and finally returning to the depot. One must actually solve a number of TSP problems, one for each available vehicle, typically after a partitioning phase in which subset of locations are assigned to a same vehicle.

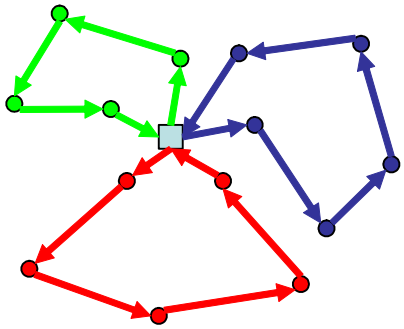

Figure 4.4: A vehicle routing for three vehicles

- *Genome Sequencing.* Very briefly, the problem consists in finding a suitable sequencing of DNA segments (*markers*) which can be reliably detected in laboratory. Once the set M of such segments has been identified (often by different laboratories), they have to be combined to form a single sequence. A suitable distance function d must be defined, where $d(m_1, m_2)$ is a measure of the probability that m_1 and m_2 are far from each other in the sequencing. We want to find a most likely sequencing, so that the sum of the distances of neighboring segments is as small as possible. To this end, we define an undirected complete graph $G(M, E)$, with costs $c_{m_1,m_2} = d(m_1, m_2)$ for $m_1, m_2 \in M$, $m_1 \neq m_2$. The original problem can be then formulated as the problem of finding a *Hamilton path*, that is a path going through every node exactly once. In turn, this problem can be reduced to a classical TSP problem on a new graph G', obtained from G introducing an additional node s and edges from s to all other nodes in M with zero cost. It is easy to see that a minimum cost Hamiltonian tour in G' corresponds to a minimum cost Hamilton path in G (obtained by simply dropping node s), see Figure 4.5.
- *Chip design and testing.* Printed circuit boards have a large number of holes for mounting chips or connecting layers. Such holes are made by drilling

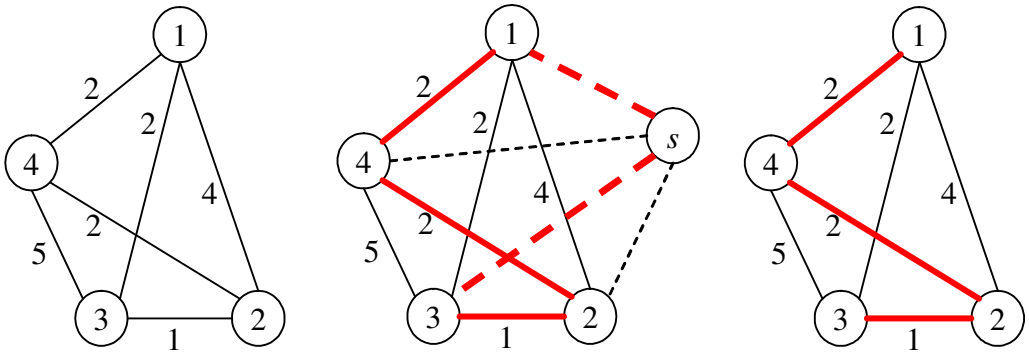

Figure 4.5: From sequencing to TSP

machines. The target here is to minimize the distance run by the drilling machine. This problem is reduced to a TSP problem by associating a node with every hole and by associating with every edge (u, v) a cost equal to the distance between hole u and hole v on the chip. A second important application is in cutting connections between logic gates on (customized) chips. This is done by a laser, which draws a Hamilton path through the gates. Minimizing the overall (Manhattan) distance run by the laser is a crucial issue for reducing production costs and times. Another important issue in chip manufacturing is testing. To do this, a number of *scan points* are established on the chip. Scan points must then be connected by a *scan chain* to allow test data to be loaded into the scan points through some input end. Clearly, the problem of finding the correct sequencing of scan points is again a TSP problem.

- *Various applications.* Several other applications concern aiming telescopes and x-rays, data mining, machine scheduling, picking items in warehouses, cutting problems in glass industry, printing circuit boards, etc.

A formulation for the TSP.

Let $G(V, E)$ be a graph, with a cost $c_e \in \mathbb{R}_+$ associated to every $e \in E$. In Chapter 1 we have shown that $x \in \{0, 1\}^E$ is the incidence vector of a Hamilton cycle in G if and only if it satisfies

$$\begin{array}{lll} \text{(i)} & x(\delta(v)) = 2 & \text{for all } v \in V; \\ \text{(ii)} & x(\delta(W)) \geq 2; & \text{for all } W \subset V,\ W \neq \emptyset,\ W \neq V; \\ \text{(iii)} & 0 \leq x \leq 1; & \end{array} \tag{4.10}$$

The constraints (i), and (iii) ensure that a solution $x \in \{0,1\}^E$ is of the form $x = \chi^F$ where $F \subseteq E$ and $d_{(V,F)}(v) = 2$ for each $v \in V$; such a set F is called a $2-$*matching* (or *2-factor*), since each node has degree 2 in the subgraph induced in G by F. Clearly, every tour is a 2-matching, so all these inequalities are valid. However, in general a 2-matching is a union of disjoint cycles, so it may not be a tour. The *2-connectivity inequalities* (ii) eliminate such a possibility, i.e., a 2-matching that satisfies (ii) is a tour (otherwise we could let W be the node set of one subtour, and we would get a violated inequality). From this it is not difficult to see that the feasible 0-1 solutions in (4.10) are precisely the incidence vectors of tours, see Problem 4.3.

2-matching

An equivalent set of constraints that can replace (4.10)(ii) is the set of *subtour elimination constraints* :

subtour elimination constraint

$$x(E[S]) \leq |S| - 1 \text{ for all } S \subseteq V,\ S \neq \emptyset,\ S \neq V. \tag{4.11}$$

These constraints were introduced in the 1954 paper by Dantzig, Fulkerson and Johnson [5]. Note that the number of 2-connectivity inequalities, or subtour elimination constraints, grows exponentially as a function of the number of nodes.

From the above discussion, it follows that the following model is a valid 0-1 linear program of the TSP:

$$\begin{array}{lll} \min & \sum_{e \in E} c_e x_e & \\ \text{subject to} & & \\ \text{(i)} & x(\delta(v)) = 2 & \text{for all } v \in V; \\ \text{(ii)} & x(\delta(W)) \geq 2; & \text{for all } W \subset V,\ W \neq \emptyset,\ W \neq V; \\ \text{(iii)} & 0 \leq x \leq 1; & \\ \text{(iv)} & x \text{ is binary.} & \end{array} \tag{4.12}$$

Relaxations.

A standard relaxation of (4.12) is obtained by removing the binary stipulation (*iv*) on the x variables. This approach and the resulting branch-and-cut will be discussed more in detail in the following sections. Here we want to mention an alternative relaxation, which is obtained by removing the 2-connectivity constraints (4.12)(ii) (but leaving the binary stipulation). This relaxation can be readily transformed into a matching problem in a bipartite graph (the assignment problem) and therefore solved efficiently. This relaxation may be combined with a suitable branch-and-bound scheme with partitioning that assures the 2-connectivity. This approach is called the *assignment problem/branch-and-bound algorithm*.

1-tree

Consider the graph G and choose a node, say node 1. A *1-tree* in G is a set $F \cup \{e, f\}$ where $(V \setminus \{1\}, F)$ is a spanning tree and e and f are two edges incident to node 1.

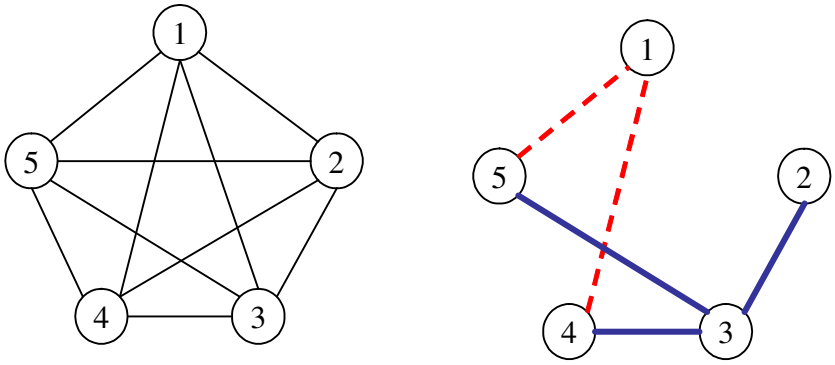

Figure 4.6: A 1-tree: the solid edges form a spanning tree F in $G[V \setminus \{1\}]$

Observe that each Hamilton tour is a 1-tree, but the converse inclusion is false. The important property of 1-trees is that they are connected, and that the following characterization of tours is valid: F is a tour if and only if it is both a 2-matching and a 1-tree. Now, the incidence vectors of 1-trees are the feasible solutions of the following set of constraints:

$$\begin{array}{lll} \text{(i)} & x(\delta(v)) = 2 & \text{for } v = 1; \\ \text{(ii)} & x(E[W]) \leq |W| - 1 & \text{for all } W \subset V \setminus \{1\},\ |W| \geq 3; \\ \text{(iii)} & x(E) = |V|; & \\ \text{(iv)} & 0 \leq x \leq 1; & \\ \text{(v)} & x \text{ is integral.} & \end{array} \tag{4.13}$$

If we here add the degree constraints $x(\delta(v)) = 2$ for $v \in V \setminus \{1\}$, we get the (incidence vectors of) tours as solutions. Instead of doing this, we relax these degree constraints using Lagrangian relaxation. We introduce Lagrangian multipliers λ_v (that are unrestricted in sign, as we have equalities) where $\lambda_1 = 0$ and get the following Lagrangian relaxation:

$$2\sum_{v \in V} \lambda_v + \min\{\sum_{uv \in E} (c_{uv} - \lambda_u - \lambda_v) x_{uv} : x \text{ is the incidence vector of a 1-tree}\}.$$

Let $v_{1T}(\lambda)$ be the optimal value of this problem. The Lagrangian dual is then (LD) $\max\{v_{1T}(\lambda) : \lambda \in \mathbb{R}^V, \lambda_1 = 0\}$ with optimal value $v(LD)$. Based on Corollary 4.6

it can be shown that z_{LD} equals the value of the linear programming relaxation of (4.12). To solve the Lagrangian subproblems one needs to find a minimum weight 1-tree. This is done by solving a minimum spanning tree problem in the subgraph $G \setminus \{1\}$ and then one finds the shortest pair of edges incident to node 1. The updating of the multipliers to solve the Lagrangian dual is done so that λ_v is increased if the degree of node v is 1 in the 1-tree, and decrease λ_v is the degree is larger than 2.

A branch-and-cut approach to solve the TSP.

As discussed in the previous section, the branch-and-cut approach consists in a branch-and-bound combined with cutting planes. There are several technical issues related to the implementation of a branch-and-cut for a particular problem. The main one is probably the possibility to *export* the valid inequalities found in a certain node of the branching tree towards other open nodes of the tree. We will not discuss this issue here, whereas we focus on separation aspects. In particular, we will describe classes of valid inequalities that are used by effective codes like the one available at [22], and in some cases we devise the corresponding separation oracles.

It is convenient to assume that G is the complete graph. This is a common technical assumption (often used in polyhedral combinatorics) which simplifies polyhedral arguments. Then the dimension of P_{TSP} is $m - n$ where $n = |V|$ and $m = |E| = n(n-1)/2$. (This is not trivial to show, but the easy thing is that $\dim(P_{TSP}) \leq m - n$ since the n degree inequalities are linearly independent.) We also assume that $n \geq 4$ (otherwise P_{TSP} is either empty or consists of one point only.)

Separating 2-connectivity inequalities.

The 2-connectivity constraints (4.12.(ii)) or their equivalent subtour elimination constraints (4.11) define facets for P_{TSP} whenever the node set W satisfies $2 \leq |W| \leq \lfloor n/2 \rfloor$, see e.g., [15]. Observe that the formulation contains an exponential number (in $|V|$) of such inequalities, so, for V sufficiently large, they cannot be represented explicitly. Thus, the dynamic simplex method must be invoked in order to solve program (4.12). Luckily, the separation of violated 2-connectivity inequalities can be performed very effectively. We describe here a separation oracle developed by Padberg and Rinaldi, see e.g., [1]. Given a point x^* we want to determine if there exists a proper subset of vertices S such that $x^*(\delta(S)) < 2$. This is an instance of the *Global Minimum Cut Problem*: given edge weights w, find a proper, non-empty subset $S \subset V$ such that $w(\delta(S))$ is minimized. In order

global minimum cut problem

to find 2-connectivity inequalities violated by x^*, we let $w_e = x^*_e$ for all $e \in E$ and find a global minimum cut S^*. If $w(\delta(S^*)) < 2$, the 2-connectivity inequality associated to S^* is violated by x^*. If $w(\delta(S^*)) \geq 2$, no 2-connectivity inequality is violated by x^* (show it).

It is immediate to realize that, given an algorithm to solve the minimum st-cut problem, a global minimum cut S^* can be easily found by the following simple algorithm:

Global Minimum Weight Cut Algorithm

Step 1. For each pair $s, t \in V$, with $s \neq t$, compute the minimum st-cut.

Step 2. A global minimum cut S^* is the minimum st-cut of minimum weight

Since the number of node pairs is $O(|V|^2)$, the complexity of the above, apparently harmless procedure is $O(|V^2| \cdot K)$, where K is the complexity of the best min st-cut solution algorithm. This can be too much in practice, even for moderate large instances. However, a simple observation can reduce the complexity by a factor $|V|$. To this end, we need first to introduce the notion of node shrinking in a graph $G(V, E)$, with edge weights w. Let $u, v \in V$, $u \neq v$. The *shrinking* of u, v in a complete graph G with weight w is a new complete graph $G'(V', E') = G|uv$ on $V' = V \setminus \{u, v\} \cup \{z\}$, with weight $w' = w|uv$ obtained from G by replacing u and v with a single node z, and by letting $w'_e = w_e$ for all $e \in E \cap E'$, and $w'_{zr} = w_{ur} + w_{vr}$ for all $r \in V \setminus \{u, v\}$. An example of shrinking is shown in Figure 4.7.

shrinking

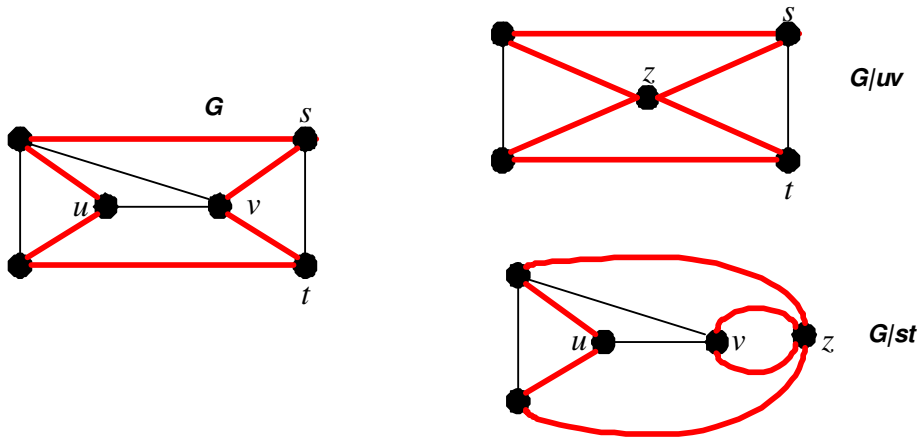

Figure 4.7: Shrinking nodes u and v (zero-weight edges are omitted)

We can easily extend the shrinking operation to set of nodes S, by simply per-

forming a sequence of standard shrinkings. Then the resulting graph, denoted by $G|S$, have a single vertex z instead of S and the new weights $w|S$ are defined as the following mapping $\phi(w, S)$:

$$\begin{aligned} (w|S)_{uv} &= w_{uv} && \text{for all } u, v \in V \setminus S \\ (w|S)_{zr} &= w(S, \{r\}) && \text{for all } r \in V \setminus S \end{aligned} \tag{4.14}$$

where, for any $w \in \mathbb{R}^E$, and sets $S, T \subset V$, $S \cap T = \emptyset$, we let $w(S,T) = \sum_{u \in S, v \in T} w_{uv}$.

Let's go back to our original task of computing a global minimum weight cut S^* in G. Consider any pair $s, t \in V$, with $s \neq t$. Then, either $s \in S^*$, $t \notin S^*$, that is S^* is an st-cut, or both s and t are in S^* (the case with S^* and $V \setminus S^*$ interchanged is equivalent and need not to be considered). Now, if $s, t \in S^*$, then it is easy to see that $S' = S^* \setminus \{u, v\} \cup \{z\}$ is a global minimum weight cut in $G|uv$ (where z is the node replacing u and v). This observation is the basis of the following algorithm to compute (the weight of) a global minimum weight cut.

Global Minimum Weight Cut Algorithm

Step 1. Let $\bar{G}(\bar{V}, \bar{E}) = G(V, E)$, $\bar{w} = w$. Let $UB = +\infty$.

Step 2. If $|\bar{V}| \leq 1$, STOP.

Step 3. Select a pair $s, t \in \bar{V}$. Find a minimum st-cut $\bar{S}$ in $\bar{G}$, with weights $\bar{w}$. If $\bar{w}(\bar{S}) < UB$, set $UB = \bar{w}(\bar{S})$.

Step 4. Set $\bar{G} = \bar{G}|st$, $\bar{w} = w|st$. Go to Step 2.

At termination, UB is the weight of a global minimum weight cut. Since at each iteration the number of nodes of the current graph decreases by one unit, the number of calls to a min st-cut solution algorithm at Step 3 is equal to $|V| - 1$. Actually the above algorithm only finds the weight of a minimum cut, but it is trivial to modify it in order to store the nodes of a minimum cut as well (by suitably handling the new nodes appearing in the shrinkings).

Safe shrinking. Shrinking is a powerful tool to effectively separating violated 2-connectivity constraints. To clarify this concept, let $G'(V', E') = G|uv$ and let $\bar{x} \in \mathbb{R}^E$. Define $\bar{x}' = \bar{x}|uv$. Suppose now that $\bar{x}$ violates a subtour elimination constraint associated to graph G. Then we may wonder if $\bar{x}'$ also violates a subtour elimination constraint associated to graph G' (in this case the shrinking is said to be *safe*). Unfortunately, this is not always the case. However, we can give sufficient

safe shrinking

conditions for this to happen, like the simple following one due to Padberg and Rinaldi (see, e.g. [1]).

Proposition 4.7. *If $\bar{x}_{uv} = 1$ and there exists a vertex t such that $\bar{x}(\{t\}, \{u, v\}) = 1$, then it is safe to shrink $\{u, v\}$.*

The above rule and many others can be applied to search for safe shrinkings in G, and shrink the corresponding edges. Also, this technique can be applied recursively to the new graphs, until no safe shrinks can be done. Most often, the final graph is considerably smaller then the original one and the minimum weight cut computations can be performed more efficiently. Finally observe that at each iteration of the global minimum weight cut algorithm we deal with a new graph (a shrinking), and safe shrinks may re-appear.

Template valid inequalities: combs.

If n is either 4 or 5, the Traveling Salesman Polytope is completely described by the trivial bounds, degree constraints and subtour elimination constraints. For larger number of nodes, other facets come into play. One such large class of inequalities is described in what follows.

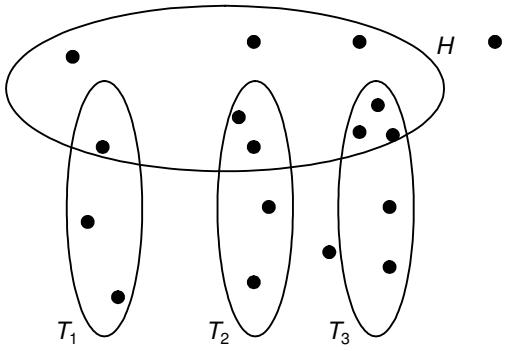

Figure 4.8: A comb

comb inequality

A *comb* in G is a class of sets H, T_i for $i \leq k$ all being subsets of the node set V and satisfying

- $H \cap T_i$ is nonempty for $i = 1, \ldots, k$;
- $T_i \setminus H$ is nonempty for $i = 1, \ldots, k$;
- the sets T_i for $i \leq k$ are pairwise disjoint;
- $k \geq 3$ is an odd number.

The set H is called the *handle* of the comb, and the T_i's are the *teeth*. Associated with each comb is a valid inequality for P_{TSP} which may be derived using the Chvátal-Gomory procedure as follows. Consider the valid inequalities

$$\begin{array}{lll} \text{(i)} & x(\delta(v)) = 2 & \text{for all } v \in H; \\ \text{(ii)} & x(E[T_i]) \leq |T_i| - 1 & \text{for } i = 1, \ldots, k; \\ \text{(iii)} & x(E[T_i \setminus H]) \leq |T_i \setminus H| - 1 & \text{for } i = 1, \ldots, k; \\ \text{(iv)} & x(E[T_i \cap H]) \leq |T_i \cap H| - 1 & \text{for } i = 1, \ldots, k; \\ \text{(iv)} & -x_e \leq 0 & \text{for } e \in \delta(H) \setminus \cup_i E[T_i]. \end{array} \tag{4.15}$$

If we add all these inequalities and divide the result by 2, we get the inequality

$$x(E[H]) + \sum_{i=1}^{k} x(E[T_i]) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - k/2.$$

Now, due to integrality since k is odd, we may round the right-hand-side down and still have a valid inequality for P_{TSP}. This gives the *comb inequality* comb inequality

$$x(E[H]) + \sum_{i=1}^{k} x(E[T_i]) \leq |H| + \sum_{i=1}^{k} (|T_i| - 1) - (k+1)/2. \tag{4.16}$$

These inequalities were introduced by Grötschel and Padberg (1979) as a generalization of the *simple comb inequalities* found by Chvátal; simple combs are combs where each $H \cap T_i$ consists of one node. It was shown by Edmonds and Johnson that the solution set of (i) the simple bound inequalities ($0 \leq x \leq 1$), (ii) the degree constraints and (iii) the comb inequalities for which each tooth has cardinality two is precisely the convex hull of the incidence vectors of 2-matchings in G. This *2-matching polytope* is actually an interesting relaxation of P_{TSP}.

The comb inequalities may be generalized into the so-called clique-tree inequalities where more handles are allowed and these are organized in a tree-like fashion, see [11]. Furthermore, the clique inequalities is a subset of the bipartition inequalities; other inequalities are called star inequalities, binested inequalities and so on. In fact, a lot of work has been done on understanding the facial structure of Traveling Salesman Polytopes and a main goal is to get a unifying understanding of the structure, not just many strange classes of inequalities.

At present no polynomial separation algorithm is known for the comb or clique inequalities. However, for a fixed number of teeth, a polynomial algorithm was found recently. There is also a polynomial algorithm for a special subclass of the comb inequalities where $|H \cap T_i| = 1$ and $|T_i \setminus H| = 1$ for each i (assuming that the point to be separated satisfies $0 \leq x \leq 1$ and all the degree constraints). This

algorithm solves the so-called minimum odd cut-set problem based on minimum cut calculations. See [16] for a description of this algorithm.

In practice, one uses different heuristics for solving the separation problems approximately (except possibly for the subtour inequalities). This means that one is not guaranteed to find violated inequalities, but those found are of course violated. This is done because it may be difficult to find exact separation algorithms, or they are complicated to implement, or, finally, they may be too slow.

Note that although the number of facets for TSP polytopes is enourmous, we only need a suitable set of m linearly independent valid inequalities to prove the optimality of a certain tour (vertex of P_{TSP}). For instance, in 120 city TSP problem originating from cities in Germany was solved to optimality by a cutting plane algorithm where 13 LP were solved and the final LP contained 36 subtour and 60 comb inequalities, see [9]. The current "world record" was a TSP problem with more than 100 000 nodes solved to optimality using a very sophisticated cutting plane algorithm (running i parallel on about 50 computers).

General Cuts.

In this final section we describe a technique to identify general violated inequalities, that is inequalities not belonging to a specific class (such as subtour elimination constraints, comb inequalities, etc.). We here give a simplified version of the presentation given in [1]. Let $\mathcal{T}$ denote the set of incidence vectors of the Hamilton tours of $G(V, E)$ and let $\bar{x} \in \mathbb{R}^E$. We want to solve the membership problem for $(\bar{x}, \text{conv}(\mathcal{T}))$, that is we want to answer the question whether $\bar{x} \in \text{conv}(\mathcal{T})$ or return an inequality violated by $\bar{x}$ but valid for $\text{conv}(\mathcal{T})$. Let A be the matrix whose columns are the elements of $\mathcal{T}$. Then $\bar{x} \in \text{conv}(\mathcal{T})$ if and only if it is a convex combination of points in $\mathcal{T}$, or equivalently if the following linear system has a solution:

$$\begin{array}{lcl} A\lambda & = & \bar{x} \\ e^T\lambda & = & 1 \\ \lambda \in \mathbb{R}_+^{\mathcal{T}} & & \end{array}$$

where e is a vector of all ones of suitable dimension.

If $\bar{x} \notin \text{conv}(\mathcal{T})$ then the above system has no solution. From Farkas lemma this happens if and only if the following system has a solution

$$\begin{array}{lcl} A^T a - eb & \leq & 0 \\ a^T \bar{x} - b & > & 0 \\ a \in \mathbb{R}^E, b \in \mathbb{R} & & \end{array} \tag{4.17}$$

Observe that the solutions to the above system are in one-to-one correspondence with the inequalities $a^T x \leq b$ which are valid for $\text{conv}(\mathcal{T})$ and are violated by $\bar{x}$.

In a branch-and-cut algorithm, where $\bar{x}$ is the optimal solution to the current relaxation, we are interested in finding inequalities which are mostly violated by $\bar{x}$, in the hope of a large improvement of the corresponding bound. To this end, we are interested in the solution to the following linear system:

$$\begin{array}{lcl} \max a^T \bar{x} - b & & \\ \quad A^T a - eb & \leq & 0 \\ \quad \sum_{e \in E} a_e & = & 1 \\ a \in \mathbb{R}^E, b \in \mathbb{R} & & \end{array} \tag{4.18}$$

Let $(\tilde{a}, \tilde{b})$ be an optimal solution to (4.18). If $\tilde{a}^T \bar{x} - \tilde{b} > 0$, then the valid inequality $\tilde{a}^T x \leq \tilde{b}$ is violated by $\bar{x}$, and is actually a most violated (by $\bar{x}$) valid inequality. Inequality $\sum_{e \in E} a_e = 1$ is included to *normalize* a. In fact, if $(\tilde{a}, \tilde{b})$ is a solution to (4.17), then $(k\tilde{a}, k\tilde{b})$ is also a solution to (4.17), for any $k > 0$. The problem (4.18) has an exponential number of rows. We apply the dynamic simplex method to solve it. To this purpose, we start with an initial small set of rows $A^T_{\mathcal{Q}}$ of A^T corresponding to a restricted set of incidence vectors $\mathcal{Q} \subseteq \mathcal{T}$ of Hamilton tours and solve the associated restricted problem. Let $(\bar{a}, \bar{b})$ be the optimal solution. We want to constraint of (4.18) (associated to a row of A^T) which is violated by $(\bar{a}, \bar{b})$, or prove that none exists. This is equivalent to solving the following TSP:

$$\max\{\bar{a}^T y : y \in \mathcal{T}\} \tag{4.19}$$

Denoting by y^* the optimal solution to the above problem, if $\bar{a}^T y^* > \bar{b}$, then we add y^* to $\mathcal{Q}$ and proceed to solve once again the restricted Problem (4.18). Otherwise we are done and $\bar{a}^T x \leq \bar{b}$ is a valid inequality for $\text{conv}(\mathcal{T})$.

Remark that all above arguments can be applied to $\mathcal{T}$ being a set of feasible solutions to any combinatorial optimizationproblem.

Even if in principle this technique may be used to generate any valid inequality for $\text{conv}(\mathcal{T})$, and solve in this way the TSP, it is apparent that we cannot apply

it directly to G. Indeed, Problem (4.19) is already as difficult as the original TSP! However, this methodology is widely used (also in different contexts) for generating *local cuts*.

Let us briefly discuss how it is applied to the solution of the TSP, in particular in [1]. The idea is to map the original set V into some smaller set $\bar{V}$, for example by shrinking. This mapping induces a linear mapping $\phi(x) : E(V) \to E(\bar{V})$, namely (4.14). One can show that, if x is the incidence vector of a Hamilton tour in G, then $\bar{x} = \phi(x)$ is an integer vector representing a *graphical Hamilton tour*, that is a tour which passes through every node a strictly positive and even number of times (the same edge can appear several times).

graphical Hamilton tour

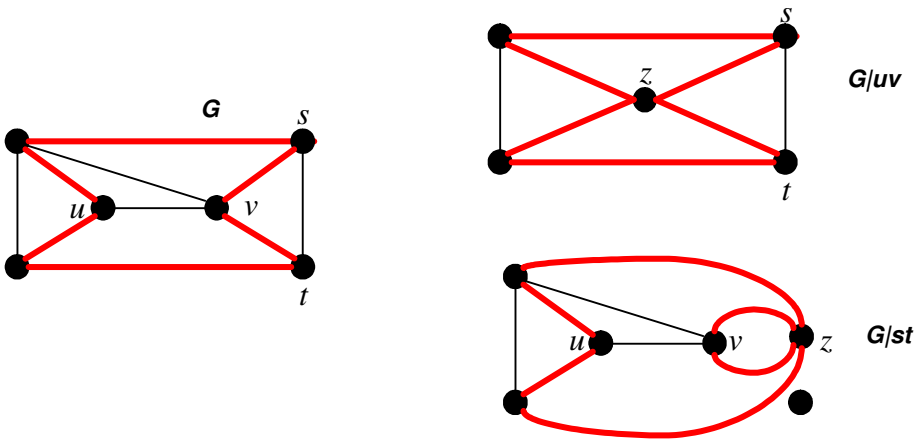

Figure 4.9: Shrinking edge uv (st) maps an Hamilton tour of G into a graphical tour of $G|uv$ ($G|st$),

In Figure 4.9, shrinking edge uv maps the original Hamilton tour into the graphical tour of $G|uv$, while shrinking edge st produces the graphical tour of $G|st$. Let us denote by $\bar{\mathcal{T}}$ the set of integer vectors representing graphical Hamilton tours of the complete graph $\bar{G}(\bar{V}, \bar{E})$ on $\bar{V}$. Consider now a point $x^* \in \mathbb{R}^E$ and its mapping $\phi(x^*) \in R^{\bar{E}}$. If $\bar{V}$ is reasonably small, we can generate a general cut which is valid for $\text{conv}(\bar{\mathcal{T}})$ and violated by $\phi(x^*)$. Then this cut can be "lifted" into the original space to produce a valid inequality violated by x^*. There are several, complicated issues which have to be addressed in this process. First, we need to ensure that, if $x^* \notin \text{conv}(\mathcal{T})$, then $\phi(x^*) \notin \text{conv}(\bar{\mathcal{T}})$ (that is, the shrinking is safe for the general cut). Second, we need a way to lift up the violated inequality into the original space. Third, we would like such inequalities to be facet defining for $\text{conv}(\mathcal{T})$. For a detailed discussion of all these issues, see [1].

4.5 Exercises

Exercise 4.1. *Consider the knapsack problem max* $\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} a_j x_j \leq b,\ 0 \leq x \leq 1\}$ *where all the data are positive integers. Define the knapsack relaxation polyope by* $P = \{x \in \mathbb{R}^n : \sum_{j=1}^{n} a_j x_j \leq b,\ 0 \leq x \leq 1\}$. *Assume that the variables have been ordered such that* $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$. *Try to guess an optimal solution and prove the optimality by considering the dual problem. Use this result to characterize all the vertices of* P. *What about the cover inequalities in relation to these vertices?*

Exercise 4.2. *Consider the branch-and-bound algorithm. Consider a node* u *in the enumeration tree with* $v(R(u)) > v(Q)$ *where* Q *is the integer program and* $R(u)$ *is the LP relaxation in node* u. *Can we prune node* u*?*

Exercise 4.3. *Prove, in detail, that (*4.12*) is a valid integer linear programming formulation of the TSP problem. Then do the same for the model obtained by replacing the cut inequalities by the subtour inequalities.*

Exercise 4.4. *Try to figure out what the odd cover inequalities might be based on the example given for the set covering problem.*

Exercise 4.5. *Consider the degree-constrained spanning tree problem. Find a valid integer linear programming formulation of this problem.*

Exercise 4.6. *Consider the following problem. We shall decide location of sevice centers among a finite set of possible locations* I. *There is given a (finite) set* J *of customers, and each shall be connected to exactly one service centre. The cost of building a service centre at location* $i \in I$ *is* c_i *and the cost of connecting customer* j *to centre location* i *is* $d_{i,j}$. *The simple plant location problem is to decide in which locations service centres should be built and the connection of customers to centres so as to minimiize the total cost (design + connection cost). Model this problem as an integer linear programming problem. Figure out some data for a small example and solve the LP relaxation as well as the ILP on a computer using an optimization package (e.g., CPLEX).*

Exercise 4.7. *Consider again the simple plant location problem from the previous problem. Suggest a Lagrangian relaxation algorithm for this problem. Discuss its properties (e.g., integrality).*

Exercise 4.8. *Develop some simple heuristics for the simple plant location problem.*

Bibliography

[1] D. Applegate, R.E. Bixby, V. Chvátal and W. Cook. *The Travelling Salesman Problem*. Pricenton University Press, 2006.

[2] R. K. Ahuja, T. L. Magnanti and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, New Jersey, 1993.

[3] N. Christofides, Worst-case anlysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, Carnagie Mellon University, Pittsburgh, 1976.

[4] G. Cornuejols and A. Sassano. On the 0, 1 facets of the set covering polytope. *Mathematical Programming*, 43:45–55, 1989.b

[5] G.B. Dantzig and D.R. Fulkerson and S.M. Johnson, Solution of a large-scale traveling-salesman problem, *Operations Research*, 7:58–66, 1954.

[6] G. Dahl, An introduction to convexity, *Lecture Notes*, University of Oslo, 2009.

[7] J. Edmonds, Matroids and the greedy algorithm, *Mathematical Programming*, 1:127-136, 1971

[8] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP- completeness*. W.H. Freeman and company, 1979.

[9] M. Grötschel. On the symmetric traveling salesman problem: solution of a 120-city problem. *Mathematical Programming Study*, 21:61–77, 1980.

[10] M. Grrötsche, L. Lovász and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer, 1988.

[11] M. Grötschel and W.R. Pulleyblank. Clique tree inequalities and the symmetric travelling salesman problem. *Mathematics of Operations Research*, 11:537–569, 1986.

[12] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, *Operations Research* 21:498–516, 1973

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem*. Wiley, 1985.

[14] L. Lovász, M.D. Plummer. *Matching Theory*. North-Holland, 1986.

[15] G.L. Nemhauser, L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, & Sons, 1988.

[16] M.W. Padberg and M. Grötschel. Polyhedral computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The traveling salesman problem*, chapter 9, pages 307–360. Wiley, 1985

[17] M. Padberg and G. Rinaldi. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Review, 33:60–100, 1991.

[18] W.R. Pulleyblank, Polyhedral combinatorics, In Nemhauser et al., editor, *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*, chapter 5, pages 371–446, North-Holland 1989.

[19] V.I. Sarvanov, N.N. Doroshko, Approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time, *Software: Algorithms and Programs*, Vol. 31, Mathematics Institute of the Belorussia Academy of Science, Minsk, 1981, 11–13 (in Russian).

[20] A. Schrijver. *Theory of linear and integer programming*. Wiley, Chichester, 1986.

[21] Alexandre Schrijver, A course in combinatorial optimization, *Lecture Notes*, CWI Amsterdam, 2009.

[22] The Travelling Salesman Problem Page, http://www.tsp.gatech.edu//

[23] Laurence Wolsey, Integer Programming, Wiley, 1998.

Index