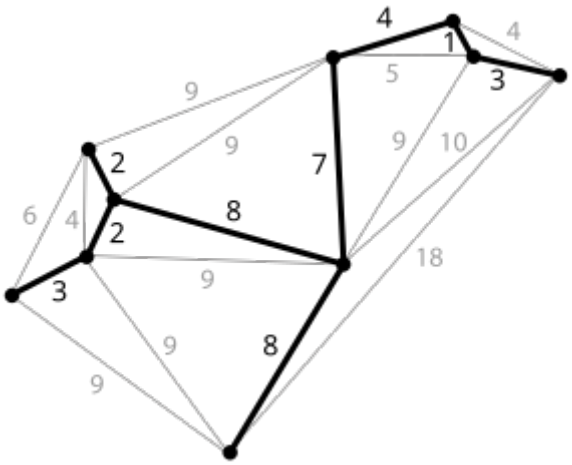

LINMA2450 Combinatorial Optimization

Simon Desmidt

This summary may not be up-to-date, the newer version is available at this address:
https://github.com/SimonDesmidt/Syntheses

Academic year 2024-2025 - Q1

UCLouvain

Contents

1. Introduction

1.1 Continuous vs Combinatorial optimization

A continuous optimization problem is expressed as

$$\max / \min f(x) \tag{1.1}$$

such that $x \in \Omega$, where $f : \mathbb{R}^n \to \mathbb{R}$ and $\Omega \subseteq \mathbb{R}^n$, where Ω does not have isolated points.
A combinatorial optimization problem has the additional constraint that

$$x \in \Omega \cap \mathbb{Z}^n$$

which can be a finite set of values. If finite, it is not solvable by exhaustive search, as it can contain loads of points even for small scale problems.

1.2 Upper/Lower bound

Let $f^* = \max\{f(x) : \quad x \in \Omega \cap \mathbb{Z}^n\}$ be the optimal value for a given problem.

- Any lower bound for f^* is called primal bound. One way to get primal bounds is to find feasible points, because if $x \in \Omega \cap \mathbb{Z}^n$, then $f_L = f(x) \leq f^*$.
- Any upper bound for f^* is called dual bound. One way to get this upper bound is to use relaxation :

  $$f^* \leq \max_x\{f(x) : \quad x \in \Omega\} = f_U$$

  Thus if $f_U - f(x) \leq \varepsilon$, we have a certificate that x is an ε-approximate solution of 1.1.

2. Knapsak problem

Suppose that we have the following :

- a bag;
- a set of objects that we want to put in the bag;
- each object has a value.

The objective of this problem is to select the objects to put in the bag in order to maximze the total value of the objects in the bag, such that the objects fit in the bag. We can only put nonnegative integer amounts of each object in the bag.

2.0.1 Mathematical formulation

- b : the volume of the bag
- n : the number of types of objects
- $0 < a_j \leq b$: the volume of one unit of object j
- $c_j > 0$: the value of one unit of object j
- x_j : the amount of object j that is put in the bag (VARIABLES)

The Linear Integer Programming Problem expression here is

$$\max \sum_{j=1}^{n} x_j c_j \quad \text{such that} \quad \begin{cases} a^T x \leq b \\ x_j \in \mathbb{N} \quad \forall j = 1, \ldots, n \end{cases} \tag{2.1}$$

Let us now suppose that

$$\frac{c_1}{a_1} \geq \frac{c_2}{a_2} \geq \cdots \geq \frac{c_n}{a_n} \tag{2.2}$$

The greedy approach is to put in the bag the maximum amount possible of object 1, then 2, and so on. The amount of object j in the bag will therefore be

$$x_j^{(g)} = \left\lfloor \frac{b - \sum_{i=1}^{j-1} a_i x_i^{(g)}}{a_j} \right\rfloor \quad \forall j = 1, \ldots, n \tag{2.3}$$

and the feasible point is $x^{(g)}$ such that it j component is $x_j^{(g)}$ calculated above.

Theorem

Let $x^{(g)}$ be the feasible point to problem 2.1 given by the equation 2.3, and suppose that the assumption 2.2 is true. Then,

$$f\left(x^{(g)}\right) \geq 1/2 f^*$$

that is, the Greedy Heuristics to a Knapsak problem gives a feasible point with a function value equal to at least 50% of the optimal value.

3. Relaxation

We will here study problems such that the relaxation of the integer constraint still gives the same solution, i.e. solving the discrete or continuous problem is equivalent.

3.1 Linear problems

3.1.1 Definitions

- A matrix $B \in \mathbb{Z}^{k\times k}$ is unimodular when it is invertible and $B^{-1} \in \mathbb{Z}^{k\times k}$.
- $B \in \mathbb{Z}^{k\times k}$ is unimodular iff $\det B \in \{-1, 1\}$.
- $A \in \mathbb{Z}^{m\times n}$ is totally unimodular (TU) when every invertible submatrix of A is unimodular.
- Let $A \in \mathbb{Z}^{m\times n}$. If $\det(B) \in \{-1, 0, 1\}$ for every square submatrix of A, then A is TU.

3.2 Resolution

A linear problem of optimization can be written under the following general form :

$$\max f(x) = c^T x \qquad \text{such that } Ax \leq b \tag{3.1}$$

We can use the simplex method in that problem, stating that the solution is one of the vertices of the feasible set (a polygon here). It is possible to rearrange the rows of A and b in such a way that we get

$$\left(\frac{B}{C}\right) \qquad \left(\frac{b_{(1)}}{b_{(2)}}\right) \tag{3.2}$$

With that, the solution $x^* = \begin{pmatrix} x^*_{(1)} \\ x^*_{(2)} \end{pmatrix}$ verifies

$$\begin{cases} Bx^*_{(1)} = b_{(1)} \\ x^*_{(2)} = 0 \end{cases} \tag{3.3}$$

B being non singular.

Theorem

Let $A \in \mathbb{Z}^{m\times n}$ be a matrix such that

1. $A_{ij} \in \{-1, 0, 1\}$;
2. Every column of A has at most 2 nonzero entries:
3. There exists a partition[a] $I_1 \cup I_2 = \{1, \ldots, m\}$ such that, if the jth column of A has exactly 2 nonzero entries, then $\sum_{i\in I_1} A_{ij} = \sum_{i\in I_2} A_{ij}$;

Then A is TU.

[a] $I_1 \cap I_2 = \emptyset$ and $I_1 \cup I_2 =$ the set.

4. Applications

4.1 Maximum Matching Problem

4.1.1 Definitions

- Given a graph $G = (V, E)$, a matching of G is a subset of edges $E' \subset E$ such that for every vertex $v \in V$, there exists at most one edge $e \in E'$ that is incident to it.
- Assume that $V = \{v_1, \dots, v_m\}$ and $E = \{e_1, \dots, e_n\}$. Given $E' \subset E$, we can then associate it to the vector $x' \in \{0, 1\} \subset \mathbb{Z}^n \subset \mathbb{R}^n$ given by

$$x'_j = \begin{cases} 1 \text{ if } e_j \in E' \\ 0 \text{ otherwise} \end{cases} \tag{4.1}$$

- Then, with this notation, we have

$$|E'| = \mathbb{1}^T x' \tag{4.2}$$

- Let us define the vertex-edge incidence matrix of $G = (V, E)$ as the matrix $M \in \mathbb{Z}^{m \times n}$ given by

$$M_{ij} = \begin{cases} 1 \text{ if } e_j \text{ is incident to } v_j \\ 0 \text{ otherwise} \end{cases} \qquad i = 1, \dots, m \qquad j = 1, \dots, n \tag{4.3}$$

4.1.2 Unweighted Problem

The maximum unweighted matching problem is to find a subgraph of a bipartite graph for which the maximum number of nodes are related to max one other node. It can be expressed in an combinatorial optimization manner such as:

$$\max \mathbb{1}^T x \qquad \text{such that } Mx \leq \mathbb{1}_m \qquad x \in \{0, 1\}^n \tag{4.4}$$

We want to be able to use the theorem of section 3.2.

Theorem

If $G = (V, E)$ with $|V| = m$ and $|E| = n$ is a bipartite graph with no self-loop[a], then its vertex-edge incidence matrix $M \in \mathbb{Z}^{m \times n}$ is a TU matrix.

[a] Edge from a node to itself.

4.1.3 Weighted Problem

Let w_j be the weight associated to the edge $e \in E$. The function to optimize then is $w^T x$, and the problem becomes

$$\max_{x \in \mathbb{R}^n} w^T x \text{ such that } Mx \leq \mathbb{1}_m \qquad x \in \{0, 1\}^n \tag{4.5}$$

4.2 Minimum Vertex Cover Problem

Definition 4.1. Given a graph $G = (V, E)$, a vertex cover of G is a subset $V' \subset V$ such that for every edge $e = \{u, v\} \in E$, we have $u \in V'$ or $v \in V'$.

The minimum vertex cover problem consists in minimizing the cardinality of $V' \subset V$ such that V' is a vertex cover of G. We define the vector $u \in \mathbb{R}^m$ such that

$$u_i = \begin{cases} 1 \text{ if } v_i \in V' \\ 0 \text{ otherwise} \end{cases} \qquad i = 1, \dots, m \tag{4.6}$$

The algebraic formulation of the problem is the following:

$$\min_{u \in \mathbb{R}^m} \mathbb{1}^T u \text{ such that } M^T u \geq \mathbb{1}_n \qquad u \in \{0, 1\}^m \tag{4.7}$$

If the graph is bipartite, then the incidence matrix M is TU, and the problem (4.7) is equivalent its relaxation:

$$\min_{u \in \mathbb{R}^m} \mathbb{1}^T u \text{ such that } M^T u \geq \mathbb{1}_n \qquad u \in [0, 1]^m \tag{4.8}$$

Given a solution u^* of that problem, the desired vertex cover V^* will be the set $V^* = \{v_i : u_i^* = 1\}$.

4.3 Shortest Path Problem

Definition 4.2. A directed graph is a pair $G = (V, A)$ where $V \neq \emptyset$ is the set of vertices and $A \subset V \times V$ is the set of arrows. If $(u, v) \in A$, the edge leaves u and arrives at v.

Let $G = (V, A)$ be a directed graph with $V = \{v_1, \dots, v_m\}$ and $A = \{a_1, \dots, a_n\}$. Then, the vertex-arrow incidence matrix $M^d \in \mathbb{R}^{m \times n}$ is defined such that

$$M_{ij}^d = \begin{cases} -1 \text{ if } a_j \text{ leaves } v_i \\ 1 \text{ if } a_j \text{ arrives at } v_i \\ 0 \text{ otherwise} \end{cases} \tag{4.9}$$

Definition 4.3. A path on $G = (V, A)$ from vertex s to vertex t is a set of arrows $P \subset A$ such that

$$P = \{(\tilde{v}_1, \tilde{v}_2), (\tilde{v}_2, \tilde{v}_3), \dots, (\tilde{v}_{N-1}, \tilde{v}_N)\} \tag{4.10}$$

where $\tilde{v}_1 = s$, $\tilde{v}_N = t$ and $\tilde{v}_i \neq \tilde{v}_\ell$ if $i \neq \ell$[1].

Given a path $P \subset A$, we can associate it to a vector $x \in \mathbb{R}^n$ such that

$$x_j = \begin{cases} 1 \text{ if } a_j \in P \\ 0 \text{ otherwise} \end{cases} \tag{4.11}$$

Let c_j be the cost associated to arrow a_j. Then, the shortest path problem can be formulated as

$$\min_{x \in \mathbb{R}^n} c^T x \text{ such that } M^d x = b \qquad x \in \{0, 1\}^n \tag{4.12}$$

where b is a vector such that its first component is -1, its last is 1, and all others are zero. The condition $M^d x = b$ translates the fact that all arrows of the path must be next to each other, and that the path starts in s and ends in t.

[1] i.e. there is no cycle.

Theorem

Let $G = (V, A)$ be a directed graph with $|V| = m$ and $|A| = n$. Denote by $M^d \in \mathbb{R}^{m \times n}$ the vertex-arrow incidence matrix. If the graph has no self-loop, then M^d is TU.

This means that once again, the combinatorial problem (4.12) is equivalent to its continuous relaxation.

Property 4.4. If a matrix A is TU, then the matrices $B = \begin{bmatrix} A \\ I \end{bmatrix}$ and $C = \begin{bmatrix} A \\ -A \end{bmatrix}$ are both also TU.

This means that the condition $x \in [0,1]^n$ can always be added to the TU incidence matrix and stay TU.

4.4 Maximum Flow Problem

In this problem, we want to maximize the flow that we can send from node s to node t, respecting the conservative law, i.e. at intermediate vertices, the flow that arrives is equal to the flow that leaves.
Let us define the matrix $\tilde{M}$ defined by the following:

$$\tilde{M}_{ij} = \begin{cases} 1 & \text{if } j \in \delta^{(+)}(i) \\ -1 & \text{if } j \in \delta^{(-)}(i) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{2, \dots, m-1\}, \qquad j \in \{1, \dots, n\} \tag{4.13}$$

where

$$\begin{cases} \delta^{(-)}(i) = \{j \in \{1, \dots, n\} : a_j \text{ leaves } v_i\} \\ \delta^{(+)}(i) = \{j \in \{1, \dots, n\} : a_j \text{ arrives at } v_i\} \end{cases} \tag{4.14}$$

In matrix form, the problem is

$$\max_{x \in \mathbb{R}^n} \sum_{j \in \delta^{(-)}(i)} x_j \text{ such that } \tilde{M}x = 0 \qquad 0 \leq x_j \leq c_j \qquad \forall j = 1, \dots, n \tag{4.15}$$

The constraint containing $\tilde{M}$ is the conservation law.

4.5 Assignment Problem

Suppose that we have n tasks and m agents. Every task should be done by some agent, and every agent is responsible for one and only one task. Let h_{ij} be the "happiness" that agent i will feel by doing task j. We want to maximize the total happiness of the agents.
Let $\mathbb{R}^{n \times n}$ be the decision variable such that

$$x_{ij} = \begin{cases} 1 & \text{if the } j\text{th task is done by agent } i \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

Then the problem is formulated as

$$\max_{x \in \mathbb{R}^{n \times n}} \sum_{j=1}^{n} \sum_{i=1}^{n} h_{ij} x_{ij} \text{ such that } \begin{cases} \sum_{j=1}^{n} x_{ij} = 1 & i = 1, \dots, n \\ \sum_{j=1}^{n} x_{ij} = 1 & j = 1, \dots, n \end{cases} \tag{4.17}$$

Let us put all this in matrix form, defining the vector x_{n^2} and the matrix $A_{2n \times n^2}$ such that, for $n = 2$, we have

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{21} \\ x_{12} \\ x_{22} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \tag{4.18}$$

It is easy to show that the matrix A is TU, and thus the problem is written in the general form:

$$\max_{x \in \mathbb{R}^{n^2}} h^T x \text{ such that } Ax = \mathbb{1}_{2n} \qquad x \in \{0,1\}^{n^2} \tag{4.19}$$

4.6 Minimum Cut Problem

Definition 4.5. Given a directed graph $G = (V, A)$ and $s, t \in V, s \neq t$, an $s - t$ cut is a pair (S, T) such that $V = S \cup T$, $S \cap T = \emptyset$, $s \in S$ and $t \in T$.

Suppose that $V = \{v_1, \ldots, v_n\}$ with $s = v_1$ and $t = v_n$. Let $\tilde{A} = \{(i,j) | (v_i, v_j) \in A\}$. We define c_{ij} as the capacity of the arrow (v_i, v_j), and finally $W \in \mathbb{R}^{n \times n}$ and $u \in \mathbb{R}^n$:

$$W_{ij} = \begin{cases} 1 & \text{if } v_i \in S \text{ and } v_j \in T \\ 0 & \text{otherwise} \end{cases} \tag{4.20}$$

$$u_i = \begin{cases} 1 & \text{if } v_i \in S \\ 0 & \text{otherwise} \end{cases} \tag{4.21}$$

The Minimum $s - t$ cut problem is the problem of finding an $s - t$ cut with minimum capacity:

$$\min_{u,W} \sum_{(i,j) \in \tilde{A}} c_{ij} W_{ij} \text{ such that } \begin{cases} W_{ij} \geq u_i - u_j & \forall (i,j) \in \tilde{A}, i \neq 1, j \neq n \\ W_{1j} \geq 1 - u_j & \forall (1,j) \in \tilde{A} \\ W_{in} \geq u_i & \forall (i,n) \in \tilde{A} \\ W_{ij} \in \{0,1\} & \forall (i,j) \in \tilde{A} \\ u_i \in \{0,1\} & \forall i \in \{1, \ldots, n\} \end{cases} \tag{4.22}$$

Theorem 4.6. The Maximum $s - t$ Flow Problem and the Minimum $s - t$ Cut Problem are strongly dual to each other.

5. Submodularity

5.1 Submodular function

Definition 5.1. Given a finite set E, let $\mathcal{P}(E)$ be the set of all subsets of E. We say that a function $f : \mathcal{P}(E) \to \mathbb{R}$ is a submodular function when for all subsets $A \subset B \subset E$ and for all $x \in E \setminus B$, we have

$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B) \tag{5.1}$$

f is said to be monotone if $f(A) \leq f(B)\ \forall A \subset B$.

Given a monotone submodular function $f : \mathcal{P}(E) \to \mathbb{R}_+$ and an integer $k \geq 1$, consider the problem of maximizing f subject to a cardinality constraint:

$$\max_{A \subset E} f(A) \text{ such that } |A| \leq k \tag{5.2}$$

Algorithm 1 Greedy Method for Problem (5.2)

1: **Step 0:** Set $S_0 = \emptyset$ and $i := 0$.
2: **Step 1:** If $i = k$, stop.
3: **Step 2:** Set

$$x_{i+1} = \arg\max_{x \in E \setminus S_i} f(S_i \cup \{x\}) \tag{5.3}$$

4: **Step 3:** Set $S_{i+1} = S_i \cup \{x_{i+1}\}$ and $i := i + 1$ and go back to Step 1.

Theorem

Let $\{S_i\}_{i=0}^k$ be generated by Algorithm 1. If S^* is a solution of (5.2), then

$$f(S^*) - f(S_i) \leq \left(1 - \frac{1}{k}\right)^i f(S^*) \qquad \forall 0 \leq i \leq k \tag{5.4}$$

This means that

$$f(S_k) \geq \left(1 - \frac{1}{e}\right) f(S^*) \geq 0.63 f(S^*) \tag{5.5}$$

5.2 k-Medoids Problem

Given a dataset $E \subset \mathbb{R}^n$ finite, find a subset $M \subset E$ with at most k elements (called medoids) such that the sum of pairwise distances between medoids and elements of E is minimized:

$$\min_{M \subset E} L(M) \equiv \frac{1}{|E|} \sum_{v \in E} \min_{e \in M} \|e - v\| \text{ such that } |M| \leq k \tag{5.6}$$

This is equivalent to

$$\max_{M \subset E} f(M) \equiv L(\{e_0\}) - L(M \cup \{e_0\}) \text{ such that } |M| \leq k \tag{5.7}$$

where e_0 is a phantom medoid. Practically, we have a dataset and find two medoids using Algorithm 1. With the above problem, we find the clusters of the dataset by assigning the points to the closest medoid.

6. Dynamic Programming

6.1 Formal description

Let p^* be the problem instance that we want to solve. Dynamic programming supposes that we can find a finite set $\mathbb{P}$ of instances p with optimal solutions S_p, with corresponding costs $C(S_p)$, such that

- $p^* \in \mathbb{P}$: the relevant problem belongs to the set.
- There is an order relation on $\mathbb{P}$: $q < p$ if q is "easier" than p. The order can be partial (not defined for all p, q). The problem p has a (possibly empty) set of immediate predecessors $R_p \subseteq \{q : q < p\}$ among the more basic problems.
- If the problem p has no predecessors, i.e. $R_p = \emptyset$, then its optimal solution S_p can be "easily" computed.
- If the problem has predecessors, then the optimal solution S_p and its cost $C(S_p)$ satisfies

$$C(S_p) = \min\{\min_{q \in R_p} C(f_{p,q}(S_q)),\ C(S_p^0)\} \tag{6.1}$$

where $f_{p,q}(S_q)$ is a way of building a solution of p by extending the optimal solution of q[1], and S_p^0 represents a "trivial" solution for p.

Example 6.1. In the case of the shortest paths from the source s to node v, the set $\mathbb{P}$ of problems consist of computing the shortest paths of length at most k ($k = 0, \ldots, n$) from the source s to every node i, and the problem order relation can be defined by $D_k(i) < D_{k'}(i')$ if $k < k'$, i.e. the max length is shorter. The set of immediate predecessors R_p of $p = D_k(i)$ is the set of $D_{k-1}(j)$ for all j from which i can be reached.

Algorithm 2 Dynamic Programming

1: Solve all problems p for which $R_p = \emptyset$.
2: **while** p^* not solved **do**
3: Pick p such that all problems of R_p are solved;
4: Solve p using (6.1).
5: **end while**

6.2 0-1 Knapsak Problem

We consider the 0-1 Knapsak problem:

$$z = \max \sum_{j=1}^{n} c_j x_j \text{ such that } \begin{cases} \sum_{j=1}^{n} a_j x_j \leq b \\ x \in [0,1]^n \end{cases} \tag{6.2}$$

where the coefficients $a_1, \ldots, a_n$ and b are positive integers. As set $\mathbb{P}$, we will define problems with smaller costs and smaller sets of objects: let λ take values from 0 to b as state, and the

[1] It is supposed to be easy to compute.

subset of variables $x_1, \ldots, x_r$ represented by r as stage, leads us to define the problem $P_r(\lambda)$ and optimal value function $f_r(\lambda)$ as follows:

$$f_r(\lambda) = \max \sum_{j=1}^{r} c_j x_j \text{ such that } \begin{cases} \sum_{j=1}^{r} a_j x_j \leq \lambda \\ x \in [0,1]^r \end{cases} \tag{6.3}$$

If $x_r = 0$, we have $f_r(\lambda) = f_{r-1}(\lambda)$. If $x_r = 1$, then $f_r(\lambda) = c_r + f_{r-1}(\lambda - a_r)$. Thus we have the following instantiation of (6.1):

$$f_r(\lambda) = \max\{f_{r-1}(\lambda),\ c_r + f_{r-1}(\lambda - a_r)\} \tag{6.4}$$

for a set of two predecessors of $f_r(\lambda)$ defined by $f_{r-1}(\lambda - a_r)$ and $f_{r-1}(\lambda)$, and the operation $f_{q,p}$ consisting in either adding the object r in the first case and keeping the solution $f_r(\lambda)$ as it is in the second one.

$\rightarrow$ Note: The Knapsak problem is NP-hard, as its complexity is exponential.

6.3 Optimal subtree of a tree

The optimal subtree of a tree problem involves a tree $T = (V, E)$ with a root $r \in V$ and weights c_v (can be negative) for $v \in V$. The problem is to choose a subtree of T rooted at r of maximum weight, or the empty tree if there is no positive weight rooted subtree. The cost function is

$$C(T(v)) = \max\{0,\ c_v + \sum_{w \in N^+(v)} C(T(w))\} \tag{6.5}$$

with $N^+(v)$ the successors of v. We can therefore define $R_{T(v)}$ by saying that computing $T(w)$ is a predecessor of the problem of computing $T(v)$ if $w \in N^+(v)$. The problem $T(v)$ for the leaf nodes have no predecessor, but are easy to compute: $T(v) = \{v\}$ if the benefit is positive and 0 otherwise.

7. Branch and bound

A branch-and-bound consists in applying a divide-and-conquer strategy on a tree enumarating all solutions. Suppose we can find a primal lower bound $\underline{z}$ and a dual upper bound $\bar{z}$ for the optimal value $z = \max_{x \in S} f(x)$. If these bounds are equal or close enough, we consider the problem solved. Otherwise, the first principle is to seek a better approximation of z by splitting the feasible set into two parts S_1, S_2 or more, and computing the bounds on those two sets, in hope that the upper and lower bounds on S_1, S_2 will be more accurate than those on S.

Property 7.1. Consider the problem $z = \max_{x \in S} f(x)$. Let $S = \cup_i S_i$ be a partition of S. Suppose moreover that

$$\underline{z}_i \leq \max_{x \in S_i} \leq \bar{z}_i \tag{7.1}$$

Then there holds $\max_i \underline{z}_i \leq z \leq \max_i \bar{z}_i$.

Property 7.2.

- If $\bar{z}_i < \underline{z}_j$ for some $j \neq i$, then there is no optimal solution to the initial problem in $S_i \Longrightarrow$ we can discard S_i;
- If $\bar{z}_i = \underline{z}_j$ for some $j \neq i$, then there exists an optimal solution to the initial problem that is not in $S_i \Longrightarrow$ we can discard S_i or S_j but not both;
- If $\bar{z}_i = \underline{z}_i$, then the feasible solution leading to $\bar{z}_i$ is an optimal solution of the sub-problem over S_i.

7.1 Nodes

The active nodes $\mathcal{A}$ are the ones that could contain an optimal solution. The closed nodes are such that the problem is already solved on S_i.

	Closed	Open
Active	Store it but no further investigation	Keep investigating
Inactive	Ignore	Ignore

7.2 Branch-and-bound algorithm

0. Initialization: Compute bounds $\underline{z}$ and $\bar{z}$, set a node 0 corresponding to the set $S_0 = S$ in the set of active nodes. The invariant is $\max_{x \in \cup_i S_i} f(x) = \max_{x \in S} f(x)$.

   Main loop, stopped when there is no active non-closed node or we found a feasible solution whose cost $\underline{z}_i$ is sufficiently close to the upper bound $\bar{z}$.

1. Branching: Select an active non-closed node i and split it into $i_1, \ldots, i_n$ by partitioning S_i in non-intersecting sets $S_i = \cup_j S_{i_j}$. Add these i_j to the set of actives nodes and remove i from it. The invariant remains valid.

2. Local bounding: Compute upper and lower bounds $\bar{z}_{i_j}$ and $\underline{z}_{i_j}$ for all nodes i_j of the partition. Add any node i_j for which $\bar{z}_{i_j} = \underline{z}_{i_j}$ to the set of closed nodes. The invariant remains valid.

3. Global bounds updates: Update the general bounds by $\bar{z} = \max_{j \in \mathcal{A}} \bar{z}_j$ and $\underline{z} = \min_{j \in \mathcal{A}} \underline{z}_j$.

4. Pruning: Remove from the set $\mathcal{A}$ every node i such that $\bar{z}_i \leq \underline{z}$.

$\rightarrow$ Note: If every active node is closed, then $\underline{z} = \bar{z}$ and we have found a feasible solution that is optimal.

7.2.1 Tricks

- How to pick the active node i in step (1): Strategy is problem-dependent.
- How to compute the bounds and with which precision: for the lower bound, can try heuristic method and use $-\infty$ if fails. For upper bound, solve a relaxation that is easier.
- How to split the set S_i into subsets: If a relaxed problem was solved for the upper bound, split by ensuring all subsets exclude the optimal solution of the relaxed problem to avoid re-finding the same upper bound.

$\rightarrow$ Note: The computation of the local bound may be stopped as soon as $\bar{z}_{i_j}$ is lower than some feasible solution, because the node will be pruned in (4) anyway.

7.3 Specific branch-and-bound for IP

Suppose we have

$$z = \max\{c^T x | x \in S\} \qquad S = P \cap \mathbb{Z}^n \qquad P \text{ a polyhedra} \tag{7.2}$$

The upper bound is found by solving the linear relaxation of the problem, removing $x \in \mathbb{Z}^n$.
The lower bound is any feasible point or $-\infty$.
For IP, the branching is done in the following way:

$$\begin{cases} S_1 = \{x : x \in S, x_i \leq \lfloor x_i^* \rfloor\} = P_1 \cap \mathbb{Z}^n \\ S_2 = \{x : x \in S, x_i \geq \lfloor x_i^* \rfloor\} = P_2 \cap \mathbb{Z}^n \end{cases} \tag{7.3}$$

For only one i such that $x_i^* \notin \mathbb{Z}$.

8. Valid inequalities and cutting planes

Consider a standard linear integer program

$$\max c^T x \text{ such that } x \in \mathbb{X} \subset \mathbb{Z}_+^n \tag{8.1}$$

The convex hull of $\mathbb{X}$, $\text{conv}(\mathbb{X})$, can be defined as

- The smallest polytope whose intersection with $\mathbb{Z}_+^n$ defines $\mathbb{X}$;
- The intersection of all such polytopes;
- The set of all convex combinations of the elements $x^{(i)}$ of $\mathbb{X}$:

$$\text{conv } (\mathbb{X}) \coloneqq \left\{ x : \sum_{t=1}^{|\mathbb{X}|} \lambda_t x^{(t)} : \lambda_t \geq 0, \sum_{t=1}^{|\mathbb{X}|} \lambda_t = 1 \right\} \tag{8.2}$$

Those three definitions being equivalent. The solution of the "relaxed" linear program

$$\max c^T x \text{ such that } x \in \text{conv}(\mathbb{X}) \tag{8.3}$$

is always an element of $\mathbb{X}$, which implies that it is an actual optimal solution to the original problem.

→ Note: the computation of the convex hull is very costly and describing it requires an exponential number of linear constraints.

The goal here is to find a trade-off between obtaining polytopes as close as possible to the convex hull, and keeping their complexity low as to control the computational burden of solving the LP.

8.1 Valid inequalities

Definition 8.1. An inequality $\pi x \leq \pi_0$ is valid for a set $\mathbb{X}$ if it holds for every $x \in \mathbb{X}$.

Definition 8.2. An inequality $\pi x \leq \pi_0$ dominates an inequality $\mu x \leq \mu_0$ if

$$\{x \in \mathbb{R}_+^n : \pi x \leq \pi_0\} \subseteq \{x \in \mathbb{R}_+^n : \mu x \leq \mu_0\} \tag{8.4}$$

Definition 8.3. Given a set of inequalities $\pi^j x \leq \pi_0^j$, $j = 1, \ldots, m$, we say that $\pi x \leq \pi_0$ is redundant if there exists positive scalars u^k, $k = 1, \ldots, m$ such that $\pi x \leq \pi_0$ is dominated by

$$\left(\sum_{j=1}^{m} u^j \pi^j \right) x \leq \sum_{j=1}^{m} u^j \pi_0^j \tag{8.5}$$

8.2 Use of the valid inequalities

1. Static: one approach is to first improve the polytope description, and then to run a standard branch-and-bound algorithm. The treatment of the polytope becomes part of a preprocessing step.
2. Dynamic, cutting plane:

Algorithm 3 Cutting plane algorithm

1: Solve the LP relaxation;
2: If the LP solution x^* is integer, stop;
3: Else add a valid inequality cutting x^*, i.e. $\pi x^* > \pi_0$, but $\pi x \leq \pi_0$ for all $x \in \mathbb{X}$;

3. Dynamic, Branch and cut: when considering a node of the Branch and Bound, we perform a few iterations of the cutting plane algorithm instead of just solving an LP.

8.3 Chvatal-Gomory procedure

Suppose that a polytope P is described by $\{x \in \mathbb{R}^n_+ : Ax \leq b\}$ and that $\mathbb{X} = P \cap \mathbb{Z}^+_n$, i.e.

$$P := \{x \in \mathbb{R}^n_+ \mid \sum_{j=1}^{n} a_j x_j \leq b\} \tag{8.6}$$

the inequality being componentwise and the a_j being the columns of A. The Chvatal-Gomory procedure consists in selecting a nonnegative vector $u \in \mathbb{R}^n_+$ and applying the following steps:

1. Positive combination of constraints: by the definition of P and the nonnegativity of u, $u^T Ax \leq u^T b$ is valid for P and for $\mathbb{X}$. It can be written as

$$\sum_{j=1}^{n} (u^T a_j) x_j \leq u^T b \tag{8.7}$$

2. Rounding of coefficients: Since all x_j are nonnegative, this holds:

$$\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leq u^T b \tag{8.8}$$

3. Rounding of independent terms: this step is valid for $\mathbb{X}$ and not for P, as it exploits the integer property of x_j. As the left-hand term is an integer, we can round the right-hand term:

$$\sum_{j=1}^{n} \lfloor u^T a_j \rfloor x_j \leq \lfloor u^T b \rfloor \tag{8.9}$$

Theorem 8.4. Every inequality for $\mathbb{X}$ can be obtained by applying this procedure finitely many times from P. This means that obtaining $\text{conv}(\mathbb{X})$ can be done in finitely many times.

8.4 Gomory cutting plane

The idea is to first solve the LP relaxation, and then to find a valid inequality that excludes the optimal solution x^* of that relaxation. Let the LP be

$$\max c^T x \text{ such that } Ax = b \qquad x \geq 0 \tag{8.10}$$

$Ax = b$ consists of m constraints and $x \geq 0$ n constraints. The m equality constraints are always tight by definition. Hence there are at least $n - m$ i such that $x_i = 0$. Separating the variables according to the basis (B) and the nonbasis (NB), the equality constraint is

$$\begin{pmatrix} A_B & A_{NB} \end{pmatrix} \begin{pmatrix} x_B \\ x_{NB} \end{pmatrix} = b \iff A_B x_B + A_{NB} x_{NB} = b \tag{8.11}$$

Assuming A_B is invertible (without loss of generality),

$$x_B + \underbrace{A_B^{-1} A_{NB}}_{\tilde{A}_{NB}} x_{NB} = \underbrace{A_B^{-1} b}_{\tilde{b}} \tag{8.12}$$

and we rewrite problem (8.10) as

$$\max c^T x \text{ such that } x_i \geq 0 \qquad b_i \geq 0 \qquad x_B + \tilde{A}_{NB} x_{NB} = \tilde{b} \tag{8.13}$$

By definition of the optimum, $x^*_{NB} = 0$, $x^*_B + 0 = \tilde{b}$, and $x^*_B \geq 0$.

- If x^*_B is an integer vector, then x^* is a solution to the initial IP problem.
- If not, then there exists at least one index i such that $\tilde{b}_i$ is not integer, and the constraint

$$x_i + \sum_{j \notin B} \lfloor \tilde{A}_{ij} \rfloor x_j \leq \lfloor \tilde{b}_i \rfloor \tag{8.14}$$

  is a valid inequality that cuts x^*.
- We can reiterate the procedure with this new smaller polytope.

Theorem 8.5. Gomory's cutting plane algorithm described above converges to the optimal solution of the IP after finitely many iterations.

8.5 Knapsack based inequalities

Knapsack based inequalities are of the form

$$\sum_{j=1}^{n} a_j x_j \leq b \tag{8.15}$$

with $b \geq 0, a_j > 0$ and $x_j \in \{0, 1\}$. Any binary problem can be reformulated as such. Moreover, if a_j is not positive, the change of variables $\bar{x}_j = 1 - x_j$ makes it positive and the variables associated to $a_j = 0$ can be discarded.

Definition 8.6. A cover C for a constraint in the Knapsack form is a subset $C \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in C} a_j \geq b$. If C is a cover, then $\sum_{j \in C} x_j \leq |C| - 1$ is a valid inequality.

Definition 8.7. A cover is minimal if, for every $i \in C$, the smaller set $C \setminus \{i\}$ is not a cover.

Let C be a cover and let the extension of C $E(C) = \{k : a_k \geq a_j, \forall j \in C\}$ be the set of the indices of the coefficients larger than or equal to all those of the cover. Then

$$\sum_{j \in C \cup E(C)} x_j \leq |C| - 1 \tag{8.16}$$

is a valid inequality.

8.5.1 Constraint lifting

Suppose we have an initial valid inequality

$$\sum_{j \in D} \alpha_j x_j \leq \beta \tag{8.17}$$

for some $\alpha_j > 0$, and $D \subset \{1, \ldots, n\}$ and binary variables x. We want to add x_t to the left-hand side of this constraint with the largest possible coefficient, for some $t \notin D$, i.e. we want to solve

$$\max \alpha_t \text{ such that } \sum_{j \in D} \alpha_j x_j + \alpha_t x_t \leq \beta \tag{8.18}$$

It must be true for any value of x_t. It is true by definition for $x_t = 0$ and so we are only interested in the case $x_t = 1$. The problem becomes:

$$\beta - \alpha_t = \max \sum_{j \in D} \alpha_j x_j \text{ such that } x_j \in \{0, 1\} \; \forall j \in D \qquad \sum_{\substack{j \in D \\ j \neq t}} a_j x_j \leq b - a_t \tag{8.19}$$

which is a Knapsack problem in α_t.

$\rightarrow$ Note: Overestimating the solution of this problem still leads to a valid inequality, but with a suboptimal a_t.

9. Lagrangian Duality

9.1 Lagrangian Dual

$$z = \max c^T x \text{ such that } x \in \mathbb{Z}_+^n \qquad Dx \leq d \qquad Ax \leq b \tag{9.1}$$

where we suppose the constraints $Dx \leq d$ to be "hard". The Lagrangian dual for $u \in \mathbb{R}_+^m$ is

$$z(u) = \max c^T x + u^T(d - D^T x) \text{ such that } x \in \mathbb{Z}_+^n \qquad Ax \leq b \tag{9.2}$$

Theorem 9.1. If $u \geq 0$, then $z(u) \geq z$.

The optimal solution $x(u)$ of the Lagrangian relaxation for a given u is also the optimal for the initial integer program if

- $Dx(u) \leq d$;
- $(Dx(u))_i = d_i$ for all i for which $u_i > 0$.

The second condition means that the penalty/reward associated to the constraint do not apply, meaning that $u^T(d - Dx) = 0$.

9.2 Optimal upper bound

Let us define the smallest of the upper bounds $z_{LD} = \min_{u \in \mathbb{R}_+^m} z(u)$.

Theorem 9.2. Let $\mathbb{X} = \mathbb{Z}_+^n \cap \{x : Ax \leq b\}$. The best Lagrangian upper bound satisfies

$$z_{LD} = \max c^T x \text{ such that } x \in \text{conv}(\mathbb{X}) \qquad Dx \leq d \tag{9.3}$$

9.3 Optimization of the upper bound

We want to find z_{LD}, solution of the following problem:

$$z(u) = \max_{x \in \mathbb{X}} c^T x + u^T(d - D^T x) = \max_{t=1,\ldots,N} c^T x^t + u^T(d - Dx^t) \tag{9.4}$$

as $\mathbb{X}$ is finite and its elements are $x^1, \ldots, x^N$. This is convex optimization, with a non smooth function.

Definition 9.3. The vector γ is a subgradient of the convex function f at point x if it holds $f(y) \geq f(x) + \gamma^T(y - x)$ for every y.

$(d - Dx(u))$ is a subgradient of $z(u)$, where $x(u) \in \mathbb{X}$ is the value of the optimum x for the relaxation with vectors u. We can solve using a subgradient descent, verifying the conditions from the course LINMA2471:

Algorithm 4 Subgradient Algorithm

1: Choose initial u^0;
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad u^{k+1} = \max(u^k - h_k(d - Dx(u^k)), 0)$;
4: **end for**

where $x(u)$ is the optimal solution of IP(u) and γ_k is the stepsize. h_k verifies $\sum_k h_k \to \infty$ and $\sum_k h_k^2 < \infty$.

9.4 Generalized assignment problem

The problem is

$$\begin{aligned} z = \max \sum_{i,j} c_{ij} x_{ij} \text{ such that } \quad & \sum_j x_{ij} = 1 \qquad \forall i \sum_i x_{ij} = 1 \qquad \forall j \\ & \sum_{ij} t_{ij} x_{ij} \leq t \\ & x_{ij} \in \mathbb{Z} \\ & x_{ij} \geq 0 \end{aligned} \tag{9.5}$$

We can dualize any constraint, and this choice is not neutral. We should balance

- the tightness of the bounds obtained with the optimal u;
- the ease to compute $z(u)$ and the subgradient;
- the ease of solving the minimization $z(u)$ over u.

10. Column generation

Consider the general problem

$$\max c^T x \text{ such that } Ax \leq b \qquad x \geq 0 \tag{10.1}$$

Using the notion of basis, we have

$$A_B x_B + A_{NB} x_{NB} = b \Longrightarrow A_B x_B^* = b \tag{10.2}$$

Applying the simplex algorithm, the question is the selection of the new variable that will enter the basis. Picking variable k, we increase it by Δx_k, meaning we have to adapt x_B^* to $x_B + \Delta x_B$:

$$A_B(x_B^* + \Delta x_B) + a_k \Delta x_k = b \Longrightarrow \Delta x_B = -(A_B)^{-1} a_k \Delta x_k \tag{10.3}$$

The variation of cost is thus

$$\Delta c = (c_k - c_B^T (A_B)^{-1} a_k) \Delta x_k -: \tilde{c}_k \Delta x_k \tag{10.4}$$

We call $\tilde{c}_k$ the reduced cost. We should not pick k if $\tilde{c}_k < 0$, and if it is positive, we have to increase Δx_k until one of the variables of the basis becomes 0; this one will then leave the basis. To be efficient, we solve

$$k = \arg\max_{j \in NB} c_j - d_B^T a_j \qquad d_B^T := c_B^T (A_B)^{-1} \tag{10.5}$$

Once again, solving this problem approximately is enough if solving it exactly takes a lot of time.

10.1 Beam cutting

Suppose we have beams of length L and a demand of d_i beams of length l_i, $i = 1, \ldots, m$. We define a pattern π as a way of cutting a beam, where the element π^k is the number of sub-beams of length l_k cut into the larger beam. A pattern is feasible iff

$$\sum_k \pi_k l_k \leq L \tag{10.6}$$

The set of possible patterns is $\Pi = \{\pi^1, \pi^2, \ldots, \pi^{|\Pi|}\}$ and the problem is

$$\min \sum_{p=1}^{|\Pi|} x_p \text{ such that } x_p \in \mathbb{Z}_+ \qquad \sum_{p=1}^{|\Pi|} x_k \pi_k^p = d_k \qquad \forall k = 1, \ldots, m \tag{10.7}$$

The algorithm to use here is

1. Build some valid initial solution, possibly to the relaxed problem instead of the IP;
2. Apply simplex iterations to the problem above;
3. When a sufficiently low cost is obtained, build a valid solution to the IP by heuristic means.

The problem can be rewritten

$$\max \sum_k d_B, k\pi_k \text{ such that } \pi_k \in \mathbb{Z}_+ \qquad \sum_k \pi_k l_k \leq L \tag{10.8}$$

which is an integer Knapsack problem.

10.2 Dantzig-Wolfe

Consider the problem

$$\max c^T x \text{ such that } x \in \mathbb{X} \qquad Dx = d \tag{10.9}$$

We want to solve the relaxation $x \in \text{conv}(\mathbb{X})$. As $\mathbb{X}$ is finite, we write $x = \sum_{i=1}^{|} \mathbb{X}|\lambda_i x^i$, for some $\lambda_i \geq 0$ and $\sum_i \lambda_i = 1$. The problem then becomes

$$\begin{aligned} &\max \sum_{i=1}^{|\mathbb{X}|} (c^T x^i)\lambda_i \\ &\text{such that } \sum_{i=1}^{|\mathbb{X}|} \lambda_i x^i = 1 \\ &\lambda_i \geq 0 \\ &\sum_{i=1}^{|\mathbb{X}|} (Dx^i)\lambda_i = d \iff \begin{pmatrix} 1 & 1 & \dots & 1 \\ Dx^1 & Dx^2 & \dots & Dx^{|\mathbb{X}|} \end{pmatrix} \lambda = \begin{pmatrix} 1 \\ d \end{pmatrix} \end{aligned} \tag{10.10}$$

10.3 Decomposition

Let us decompose $X = X^1 \times X^2 \times \cdots \times X^n$, i.e. $x = (x_1, \dots, x_n)^T$, with $x_k \in X_k$ for sets k on which it is simple to optimize. We write the problem under the form

$$\max \sum_k c_k^T x_k \text{ such that } x_k \in X_k \qquad \sum_k D x_k = d \tag{10.11}$$

Using a relaxation in the convex hull of the X_k, we get

$$\begin{aligned} &\max \sum_k \sum_i (c_k^T x_k^i)\lambda_{k,i} \text{ such that} \\ &\lambda_{k,i} \geq 0 \qquad \forall i,k \\ &\sum_i \lambda_{k,i} = 1 \qquad \forall k \\ &\sum_k \sum_i (D_k x_k^i)\lambda_{k,i} = d \end{aligned} \tag{10.12}$$

And the equality constraints are under the form

$$\left(\begin{array}{cccc|cccc|c|cccc} 1 & 1 & \dots & 1 & & & & & & & & & \\ & & & & 1 & 1 & \dots & 1 & & & & & \\ & & & & & & & & \ddots & & & & \\ & & & & & & & & & 1 & 1 & \dots & 1 \\ \hline D_1 x_1^1 & D_1 x_1^2 & \dots & D_1 x_1^{|X_1|} & D_2 x_2^1 & D_2 x_2^2 & \dots & D_2 x_2^{|X_2|} & \dots & D_n x_n^1 & D_n x_n^2 & \dots & D_n x_n^{|X_n|} \end{array}\right) \lambda = \left(\begin{array}{c} 1 \\ 1 \\ \vdots \\ 1 \\ \hline d \end{array}\right)$$

The reduced cost corresponding to a variable $\lambda_{k,i}$ is

$$\tilde{d}_{B,k} + (c_k - \tilde{d}_{B,n+1:n+m} D_k) x_k^i \tag{10.13}$$

For a given k, finding the variable $\lambda_{k,i}$ with the best reduced cost consists in solving

$$\tilde{d}_{B,k} + \max_{x \in X_k} (c_k - \tilde{d}_{B,n+1:n+m} D_k) x_k \tag{10.14}$$

which is an easier problem. The decomposition of X implies a number of variables of $c \cdot n$ instead of c^n if all X_k have cardinality c.

11. Heuristic methods

→ Note: Heuristics are typically not designed to find the optimal solution and come with no such guarantee.

11.1 Local explorations

Suppose we want the solution of $\min_{x\in\mathbb{X},y\in\mathbb{R}^n} g(x,y)$. We can define the function $f(x) = \min_{y\in\mathbb{R}^n} g(x,y)$ and the problem becomes equivalent to solving $\min_{x\in\mathbb{X}} f(x)$.

11.1.1 Graph representation

We represent here the set of solution as a graph whose nodes are the solution. A solution v is a neighbour of w if v can be obtained from w by a basic transformation, the meaning of which is problem-dependent.

11.1.2 Local search

At each iteration, one will consider all neighbours of the current node v_i and pick as v_{i+1} the node for which $f(w)$ is the smallest. Another solution is to select a random neighbour w and keep it as v_{i+1} if $f(x) \leq f(v_i)$, or randomly select k neighbours and take the one with the smallest $f(w)$. One problem would be being stuck in a local minimum. A solution is to run several instances of the algorithm in parallel, or detect when the process appears to have converged to a local minimum and restart it from another point.

11.1.3 Tabu search

Tabu search consists in maintaining a list of nodes that have already been visited and preventing the algorithm from returning to these nodes. This will force the algorithm to get away from the local minima.

11.1.4 Simulated annealing

Simulated annealing allows, with a certain probability, moving towards solutions with higher cost, in such a way that there is always a positive probability of escaping any local minimum basin.
If $f(w) < f(v_i)$, then move to w. If $f(w) > f(v_i)$, the algorithm sets $v_{i+1} = w$ with probability $e^{-(f(w)-f(v_i))/T}$, where T is the temperature. T should be chosen to decrease slowly over time. A possibility to decrease T every I iterations is $T = T_0\frac{a}{a+k}$. If T decreases sufficiently slowly, simulated annealing has been proven to converge to a global minimum.

11.2 Greedy Algorithms

Working on the graph representation, there is an edge from v to w if w is larger than v and can be obtained from v by a simple "additive" operation, i.e. switching a x_i from 0 to 1 or other possibilities. Greedy algorithms consist then in starting from a node corresponding to an empty

or trivial solution, and exploring the directed network by keeping moving the out-neighbour of the current node with the lowest cost, until reaching a sink node (solution that cannot be increased).

11.3 Genetic algorithms

In genetic algorithms, the solutions are individuals. At each iteration,

- Certain individuals are selected to procreate, i.e. mixing two or more solutions to produce new ones;
- Random perturbations (mutations) are introduced to allow for more diversification;
- Each solution has a probability of being removed (killed) to keep the number of living solutions small.

For that, we need

- A fitness function: takes into account the objective value and the time the solution has lived;
- A way to merge solutions;
- A mutation function;

11.4 Accuracy of the TSP

Suppose that there exists a path p_{ij} of cost $w_{p_{ij}}$ from node i to j. Suppose also that we have built a partial solution to the TSP that ends at i and does not contain j, and we want to extend it to go to j. It is always possible to achieve this extension at cost no greater than $w_{p_{ij}}$ if

- The rules of the TSP allow going several times through the same node.
- The graph satisfies the triangular inequality (and is thus complete).

Lemma 11.1. Consider an instance of the TSP satisfying the triangular inequality. If there exists a walk W passing through every node and finishing at its starting point, then one can construct from W a salesman tour S with cost $C(s) \leq C(W)$, at a cost proportional to the length of W.

Lemma 11.2. The weight of the optimal salesman tour S^* exceeds that of the minimum weight spanning tree T^*, i.e. $C(T^*) \leq C(S^*)$. The spanning tree is easily computed and thus we have a lower bound.

If we double all the edges, then all nodes have an even degree, and there exists an Eulerian tour: $C(W) = 2C(T^*)$. Using thoses results, we have a bound on the greedy approach: $C(W) \leq 2C(C^*)$.

11.4.1 Christofidès algorithm

The method is the following:

- Build the optimal spanning tree;
- Double all the edges so as to obtain a "double tree" subgraph where all nodes have an even degree;
- Take an Eulerian tour of the double-tree subgraph;

- If possible, reduce the cost by taking shortcuts, using lemma (11.1).

Rather than doubling each edge, we can also simply increase every odd degree by one, so that each node would have an even degree. We need to add a number of edges equal to exactly half of the number of nodes with odd degrees.

Lemma 11.3. Let S^* be the optimal salesman tour on a graph with an even number of nodes and M^* the minimal weight perfect matching. Then, $C(S^*) \geq 2C(M^*)$.
On an arbitrary graph, the cost $C(S^*)$ of the optimal TSP is at least twice the cost $C(M_V^*)$ of the minimal perfect matching M_V^* on the subgraph induced by any subset V^* of an even number of nodes.

We can show that $C(S) \leq \frac{3}{2}C(S^*)$.