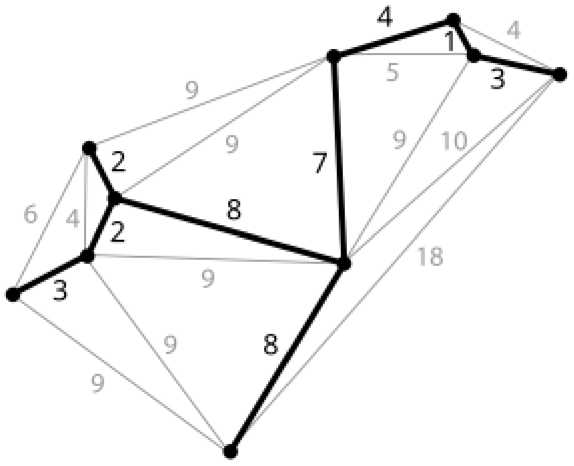

LINMA2450 Combinatorial Optimization

Simon Desmidt

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 :

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

  Thus if $f_\mathcal{U} - f(x) \leq \varepsilon$, 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

$$\begin{pmatrix} B \\ \hline C \end{pmatrix} \qquad \begin{pmatrix} b_{(1)} \\ \hline b_{(2)} \end{pmatrix} \tag{3.2}$$

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

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

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 $V = \{v_1, \ldots, v_m\}$ and $A = \{a_1, \ldots, a_n\}$. Then, the vertex-arrow incidence matrix $M^d \in \mathbb{R}^{m \times n}$ is defined such that

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

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

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

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:

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

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

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

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 Mininimum 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
       $x_{i+1} = \arg\max_{x \in E \setminus S_i} f(S_i \cup \{x\})$        (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 apths 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**
```

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

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 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.